# VIEWCACHE: An Incremental Pointer-Base Access Method for Distributed Databases

**Part I:** The Universal Index System Design Document

**Part II:** The Universal Index System Low-Level Design Document

**Part III:** User's Guide

**Part IV:** Reference Manual

**Part V:** UIMS Test Suite

**Steve Kelley**
**Nick Roussopoulos**
**Timos Sellis**

Advanced Communication Technology Inc.
1209 Goth Lane
Silver Spring, Maryland 20905

**Final Report**

**SBIR Phase II Contract Number NAS5-30628**

Prepared for
Goddard Space Flight Center
Greenbelt, Maryland 20771

**October 10, 1992**

# Part I: The Universal Index System Design Document

## PROJECT SUMMARY

Today, there is a great diversity of computers, operating systems, database management systems, and communication protocols. As a result of this heterogeneity, computer users are required to learn many different data access methods in order to obtain the information they need. This causes an attitude of "it's too much trouble to learn all these different systems," which leads to a significant amount of software and data duplication.

There are several approaches that can be taken to solve the heterogeneity problem: two of which are *standardization* and *uniformization*. *Standardization* is the concept of choosing one specific system to use, and expecting or requiring everyone to follow this standard. This, however, does not provide an adequate solution because it could be extremely costly to change to the standard if a different system was being used. *Uniformization* is the concept of creating a layer on top of current systems that provides uniform access to all data, regardless of the underlying system. This allows the underlying systems to remain unchanged, yet also provides a single common access method for users to access data.

The Universal Index System (UIS) is a system that uses uniformization to solve the heterogeneity problem among database management systems. UIS provides an easy-to-use common interface to access all underlying data, but also allows different underlying database management systems, storage representations, and access methods.

# 1. SYSTEM OVERVIEW

## 1.1. Main UIS Components

UIS is a system that manages and maintains indexes, sets, indexsets, and indexkits. An *index* is an object that associates terms with pointers. A simple example of an index is the index of a book. It associates a term used in the book with the page number(s) on which that term appears. Another example of an index is a subject index in a library catalog, which associates library books with different subjects.

A *set* is an object that contains only pointers. Usually sets are created by extracting the pointer field from an index. Using the example of a book's index, a set could be created from the index by the definition "all the page numbers that contain the words 'database', 'data model', 'data definition language', or 'data manipulation language'."

An *indexset* is a catalogued collection of indexes and sets. Every index and set must be associated with exactly one indexset. In addition to the indexes and sets belonging to an indexset, an indexset also contains an index catalog to maintain all the information for managing indexes, and a set catalog to maintain all the information for managing sets.

An *indexkit* is a logical grouping of an introduction, index, dictionary and thesaurus. The introduction component of an indexkit is an object which contains a textual description of the index. The dictionary component of an indexkit is an object that associates terms given in the index with their definition. It is used to assist the user in accessing the index. The thesaurus component of an indexkit is an object that associates terms given in the index with other terms. The thesaurus supports both generalization and specialization of terms in the index. The thesaurus is also used to assist the user in accessing the index. The introduction, dictionary and thesaurus components are neither managed nor maintained by UIS. Figure 1 shows the relationships among the different objects managed by UIS.

## 1.2. UIS Capabilities

UIS provides commands that allow the user to create and manipulate indexes, sets, indexsets, and indexkits.

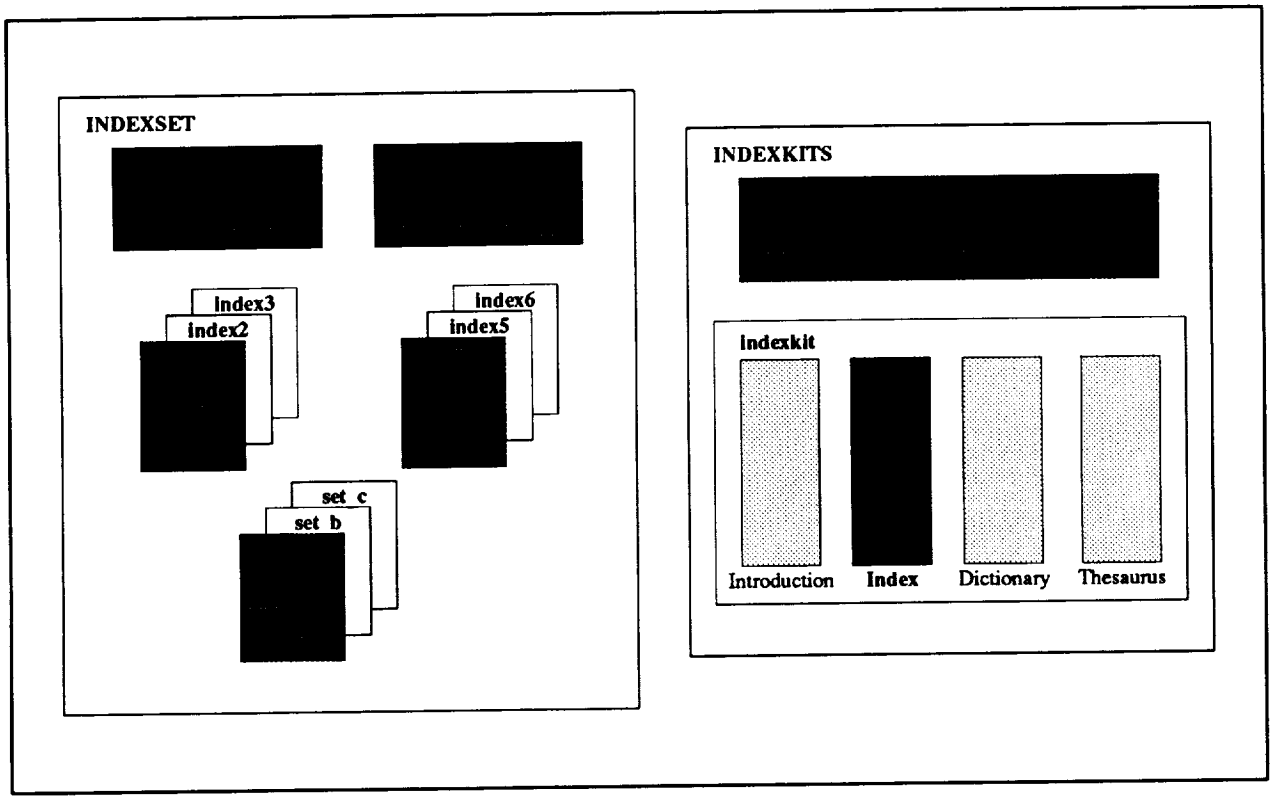


**Figure 1** - Relationships among Indexes, Indexsets, Indexkits and Kitsets.

### 1.2.1. Index Commands

UIS uses the notion of *current* objects to simplify the index commands. The user specifies which instance of an object is to be *current* i.e. to be "worked on," and then subsequent commands are performed on the *current* object. The index commands rely on the existence of a *current index*, *current index row*, and *current index boolean*.

The *current index row* is set to be the tuple in the *current index* that was most recently accessed by the navigation routines (see below for a description of the navigation routines). The *current index boolean* is a boolean condition chosen by the user to assist in navigation.

UIS provides a relationally complete set of commands for indexes. In addition to commands that allow the user to create, insert into, delete from, save and destroy indexes, there are routines that allow the user to retrieve a previously created index for either modification or read only, return an index (the opposite of retrieve) and pick an index to be the *current index*.

There are commands to allow the user to navigate both forward and backward through an index, accessing a single tuple at a time. UIS provides the user with *index booleans* and *index selects* to assist in this navigation. An *index boolean* is a boolean condition defined by the user to restrict the search to a subset of the index. For example, the user could define an index boolean, ''camseq = ''LFP1010" to restrict the search on an index to only those tuples of an IUE index having ''LFP1010'' as camera sequence number. The user can create index booleans during a user session, but they do not persist beyond the end of that session. UIS provides commands to create, modify, list (display), pick (make as current), and delete index booleans. There are also commands to allow the user to reproduce indexes. These include copying and moving an index to an indexset.

To support interfaces to programming languages, there are commands to allow the user to bind attribute values to program variables, i.e. embedding UIS commands in an application written in C. There are two commands for binding to program variables, one for binding a single attribute (column) from an index, and one that allows for binding a whole row from an index. These commands cannot be used during an interactive session.

### 1.2.2. Indexset Commands

UIS provides a few commands to manipulate indexsets. At this point a user can only create and destroy indexsets. In the future, we plan to add commands such as include copy, subset, intersect, subtract and union, and commands to copy and move indexsets.

### 1.2.3. Indexkit Commands

Although not implemented in the current prototype, several commands to manipulate indexkits have been designed for UIS. In addition to commands that allow a user to create and destroy indexkits, there are commands to allow the user to reproduce indexkits. These include copy, subset, intersect, subtract and union. Subsetting an indexkit is defined to be a new indexkit, whose components are the result of subsetting each of the components in the original indexkit. Intersecting two indexkits is defined to be a new indexkit, whose components are the result of intersecting corresponding components of the two original indexkits. Similar definitions hold for union and subtraction.

### 1.2.4. Command Summary

Tables A, B, C, and D provide a summary of the index, set, indexset and indexkit commands, respectively.

# Table A: Index Commands

| Index Management Commands | | |
|---|---|---|
| create index | drop index | insert index |
| update index | move index | delete index |

| Index Reproduction Commands | | |
|---|---|---|
| copy index | intersect index | subset index |
| subtract index | union index | |

| Index Searching Commands | |
|---|---|
| find term in index | build set with term |
| build set with list | build set with range |

| Index Browsing Commands | | |
|---|---|---|
| retrieve index | pick index | save index |
| return index | list indexes | |

| Index Navigation Commands | | |
|---|---|---|
| first in index | next in index | fetch using index |
| last in index | previous in index | |
| build index boolean | list index booleans | pick index boolean |
| modify index boolean | drop index boolean | |
| build index select | list index selects | pick index select |
| modify index select | drop index select | |

| Index Run-Time Environment Commands | |
|---|---|
| bind index column | bind index table |

# Table B: Set Commands

| Set Management Commands | | |
|---|---|---|
| build empty set | drop disk set | insert set |
| delete set | update set | |

| Set Reproduction Commands | | |
|---|---|---|
| combine sets | restrict sets | sort sets |

| Set Browsing Commands | | | |
|---|---|---|---|
| retrieve set | pick set | build empty memory set | save set |
| return set | list sets | drop set | |

| Set Navigation Commands | | |
|---|---|---|
| first in set | next in set | fetch using set |
| last in set | previous in set | |
| build set boolean | list set booleans | pick set boolean |
| modify set boolean | drop set boolean | |
| build set select | list set selects | pick set select |
| modify set select | drop set select | |

| Set Run-Time Environment Commands |
|---|
| bind set column |

## Table C: Indexset Commands

| Indexset Management Commands | |
|---|---|
| create indexset | drop indexset |
| alter indexset | move indexset |

| Indexset Reproduction Commands | | |
|---|---|---|
| copy indexset | intersect indexset | subset indexset |
| subtract indexset | union indexset | |

| Indexspace Commands | |
|---|---|
| create indexspace | alter indexspace |

## Table D: Indexkit Commands

| Indexkit Management Commands | |
|---|---|
| create indexkit | drop indexkit |
| update indexkit | move indexkit |

| Indexkit Reproduction Commands | | |
|---|---|---|
| copy indexkit | intersect indexkit | subset indexkit |
| subtract indexkit | union indexkit | |

# 2. THE DESIGN OF UIS

The development of the UIS prototype was divided into several phases: the requirements phase, the design phase, the implementation phase and the testing and integration phase. This approach was taken in an attempt to resolve any conflicts in the proposed system as early as possible.

The requirements document contains a functional description of what the system should do. The purpose of the design phase is to convert the functional description of **what** the system should do into an algorithmic description of **how** the system should do it.

The design phase primarily concentrated on two tasks. First, we had to determine what information needed to be available to the system during execution and what information needed to be available from one execution to the next (persistent information). Second, we needed to translate the functional requirements of the user commands into design specifications. These two tasks were performed in a stepwise fashion to yield a cohesive and consistent design.

## 2.1. System Information

UIS manages and maintains four different types of objects: indexes, sets, indexsets, and indexkits. In order to do so properly and efficiently, the system needs to have available certain information about each object. As an example, consider a library: how useful or efficient would a library be if it did not have a catalog that listed what books were contained in the library, or where they were located? Probably not very useful, definitely not very efficient. In the same way that a library catalogs all the **objects** that it manages, so must UIS. This section describes which information UIS needs to efficiently manage its objects.

### 2.1.1. System Catalog — Indexes

In Section 1 we defined conceptually what an index is. To determine what persistent information we need for indexes, we need to know what an index is structurally. "Structurally, an index is a table in which some of the columns are the items indexed and the last column is the pointer. An index is of type k if it has k-item tuples (columns). The format of an index depends on the internal representation of the index. Examples of formats are B-trees, R-trees, and heaps."

Given this structural definition, we see that some of the information that needs to be stored include the name of the index, its type, and its format. Other information that is necessary are the attribute or column

names, their types, lengths and their location within the tuple (offset). This information is necessary when checking whether or not a user's command is valid, and to assist the system in locating and extracting attribute values. Another piece of information used to assist the system in index manipulation and validation is the index's tuple width (the total size of the tuple). In addition, we decided it would be helpful to store whether or not a given index had an indexkit associated with it. This would allow us to remain consistent with the index-kit system information (discussed later).

Because an index can have any number of attributes, we decided it would be easier to have two system catalogs. The first one contains all the information about the index except for the attribute information. A second catalog contains the attribute information. This approach was taken to simplify the catalog access routines (if a single catalog were used, the access routines would have to support variable length entries). Figure 2 describes pictorially the system catalog information for indexes. It contains two example indexes:

**Index Catalog**

| Index Name | Index Type | Index Format | Indexkit Name | Indexkit set | Indexspace Name | Tuple Width |
|---|---|---|---|---|---|---|
| *FOLLET_EOTN* | *2* | *BTREE* | *FOLLET_EOTN* | *FOLLET* | *EARLY_WORKS* | *44* |
| *SUBJECTS* | *5* | *RTREE* | *SUBJECT_LIB* | *LIBRARY* | *PG_COUNTY* | *204* |

**Attribute Catalog**

| Index Name | Attribute Name | Attr Type | Attr Length | Attr Offset |
|---|---|---|---|---|
| *FOLLET_EOTN* | *TERM* | *STRING* | *40* | *0* |
| *FOLLET_EOTN* | *PAGE_NUM* | *INT* | *4* | *40* |
| *SUBJECTS* | *SUBJECT_TERM* | *STRING* | *40* | *0* |
| *SUBJECTS* | *AUTHOR* | *STRING* | *40* | *40* |
| *SUBJECTS* | *TITLE* | *STRING* | *100* | *80* |
| *SUBJECTS* | *ISBN_NUMBER* | *STRING* | *12* | *180* |
| *SUBJECTS* | *LC_NUMBER* | *STRING* | *12* | *192* |

Index Format =
{ BTREE, RTREE, HEAP }

Attr Type =
{ INT, FLOAT, CHAR, STRING }

**Figure 2** - System Catalog Information for Indexes.

FOLLET_EOTN (a book index for Ken Follet's *The Eye Of The Needle* and SUBJECTS (a library catalog of subjects which references books).

### 2.1.2. System Catalog — Indexsets

An indexset has several components (see Figure 3). It contains an index catalog discussed in the previous section, a set catalog, a transaction log, and then the indexes and sets themselves that belong in the indexset. The transaction log contains information about updates to the indexes and sets in the indexset. It is used in transaction management (currently unimplemented). UIS allows the user to explicitly specify all the buffer management constants needed for the management of indexset components. As a result, the system catalog information for indexsets must store all this information.

Before explaining the system catalog information for indexsets, we need to clarify what is meant by *databook* and *indexspace*. When defining an indexset, the user creates a logical space in which indexes and sets will belong at some point in the future. The *databook* objects are these logical spaces. An *indexspace* is the physical storage space on the disk that corresponds to the logical space defined by the databooks. Indexspaces can contain several databooks, and databooks can span more than one indexspace. Having the user be able to specify both logical space and physical space allows the user to place indexes physically near each other or logically near each other.

Given these new objects, an indexset is composed of the following components: index catalog, set catalog, transaction log, any number of databooks, and any number of indexspaces. For each of these components, the system needs to have information about the names of each of these components, the initial physical size of these components, their maximum size, and the rate at which these components can increase (when an insertion needs to be made and there is no space, an increase is requested and as long as the maximum size has not been reached, the increase is allowed).

Storing all this information creates a complicated system catalog structure. The databook and indexspace information for indexsets is stored in its own catalog. This is due to the fact that there can be any number of these objects in an indexset (similar to the attribute information for indexes). Since the directory, index catalog, set catalog and transaction log components are required for each indexset, and an indexset can contain at most one of each component, all of this information can be stored in a single catalog along with the indexset name. In addition, it was decided to have entries in this catalog for the total number of databooks and

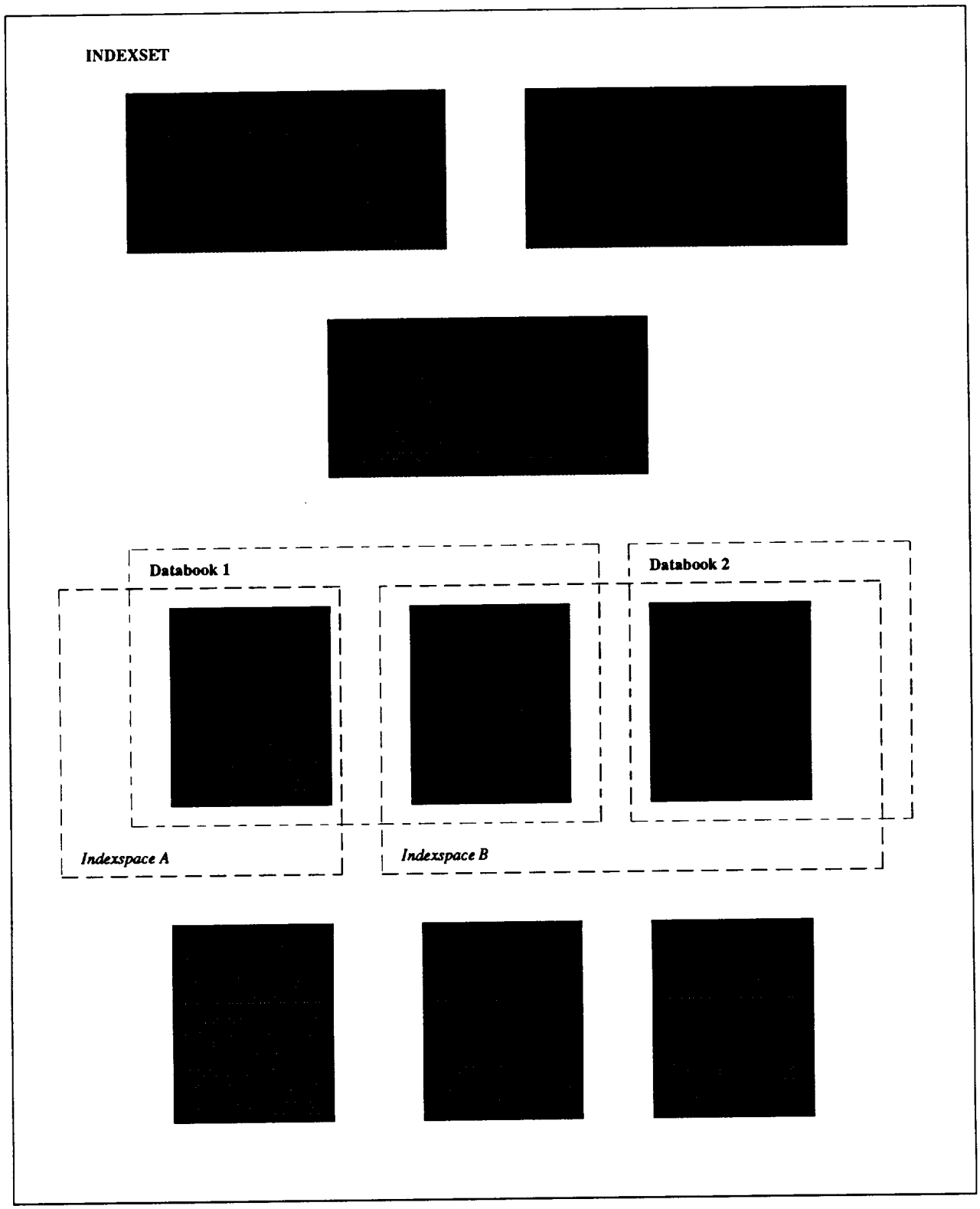


Figure 3 - Physical Structure of an Indexset

indexspaces in the indexset, to assist in retrieval from the other catalogs.

Figure 4 describes pictorially the system catalog information for indexsets. It contains two examples of indexsets: FOLLET_SET (an indexset that contains all the index information about Ken Follet's books) and SUBJECT_SET (an indexset that contains all the subject information at a specific library). For example, the index catalog component for FOLLET_SET says that the index catalog is located in the file *FOLLET_IDX*. Its initial size is 4096 bytes, and when the system needs more space for the index catalog, space is allocated in blocks of 1096 bytes. If the size of the index catalog reaches 200000, no more space will be allocated to the

**Indexset System Catalog**

| ISET_name | # databook | # indexspace | Directory Info | | Index Catalog Info | | Set Catalog Info | | Log Info | |
|---|---|---|---|---|---|---|---|---|---|---|
| *FOLLET_SET* | *1* | *1* | Device Name | *FOLLET.DIR* | Device Name | *FOLLET_IDX* | Device Name | *FOLLET_SET* | Device Name | *FOLLET_LOG* |
| | | | Initsize | *4096* | Initsize | *4096* | Initsize | *4096* | Initsize | *4096* |
| | | | Incrsize | *1096* | Incrsize | *1096* | Incrsize | *1096* | Incrsize | *1096* |
| | | | Maxsize | *200000* | Maxsize | *200000* | Maxsize | *200000* | Maxsize | *200000* |
| *SUBJECT_SET* | *1* | *1* | Device Name | *SUBJECT.DIR* | Device Name | *SUBJECT_IDX* | Device Name | *SUBJECT_SET* | Device Name | *SUBJECT_LOG* |
| | | | Initsize | *4096* | Initsize | *4096* | Initsize | *4096* | Initsize | *4096* |
| | | | Incrsize | *1096* | Incrsize | *1096* | Incrsize | *1096* | Incrsize | *1096* |
| | | | Maxsize | *200000* | Maxsize | *200000* | Maxsize | *200000* | Maxsize | *200000* |

**Databook System Catalog**

| ISET_name | Device_name | Initsize | Incrsize | Maxsize |
|---|---|---|---|---|
| *FOLLET_SET* | *EARLY_WORKS* | *4096* | *1096* | *200000* |
| *SUBJECT_SET* | *COMPUTER_SC* | *4096* | *1096* | *200000* |

**Indexspace System Catalog**

| ISET_name | Device_name | Initsize | Incrsize | Maxsize |
|---|---|---|---|---|
| *SUBJECT_SET* | *COMPUTER_SC.SPC* | *4096* | *1096* | *400000* |
| *FOLLET_SET* | *EARLY_WORKS.SPC* | *4096* | *1096* | *200000* |

**Figure 4** - System Catalog Information for Indexsets.

index catalog. The **Databook System Catalog** and the **Indexspace System Catalog** contain similar information about the databooks and indexspaces in the indexset.

### 2.1.3. System Catalog — Indexkits

As defined an Section 1, an indexkit is a logical grouping of an introduction, index, dictionary and thesaurus. In order for the system to understand this logical grouping, it needs to keep track of which instances of each component make up this logical grouping. As a result, the system information needed for each indexkit is the name of the indexkit, the introduction name and its location (intro_set), the index name and its location (indexset), the dictionary name and its location (dict_set), and the thesaurus name and its location (thes_set) (Remember that the introduction, dictionary and thesaurus components are not managed by UIS). With this information, the system can efficiently execute all the indexkit commands.

Figure 5 illustrates the system catalog information for indexkits. It contains two example indexkits (they correspond to the two index examples of Figure 2: FOLLET_EOTN (an indexkit corresponding to the index, having the same name), and SUBJECT_LIB (an indexkit corresponding to the subject index of a library catalog). Indexkits are not implemented in the current prototype.

## 2.2. Run-Time Information

In addition to persistent information about each object in the system, during execution, there is a need to track additional information about the state of objects currently being manipulated or accessed by the system. Tracking such information is essential to maintaining a consistent system. This information will be particularly crucial in a multi-user environment, when it is possible for different users to try to update the same data

**Indexkit System Catalog**

| Indexkit Name | Intro Name | Intro Set | Index Name | Index Set | Dictionary Name | Dictionary Set | Thesaurus Name | Thesaurus Set |
|---|---|---|---|---|---|---|---|---|
| *FOLLET_ EOTN* | *FOLLET_ EOTN_INTRO* | *FOLLET_ INTROS* | *FOLLET_ EOTN* | *FOLLET_ BOOKS* | *FOLLET_ DICTIONARY* | *NOVEL_ DICTS* | *FOLLET_ THESAURUS* | *NOVEL_ THES* |
| *SUBJECT_LIB* | *SUBJECTS_ INTRO* | *LIBRARY INTROS* | *SUBJECTS* | *LIBRARY* | *SUBJECT_ DICTIONARY* | *LIBRARY DICTS* | *LIBRARY THESAURUS* | *LIBRARY THES* |

**Figure 5** - System Catalog Information for Indexkits.

at exactly the same time. If the system were keeping no information about objects currently in the system, then it would have no way of preventing different users from updating the same data at the same time; there would be no way to guarantee a consistent system. This section describes what information UIS needs during execution to maintain consistency of the objects.

### 2.2.1. Run-Time Information — Indexes

As described in Section 1, the index routines support the notion of a *current* index. What this means in terms of execution, is that a user can have any number of indexes retrieved at a time (i.e. open and accessible), of which at most one may be the *current* index. We adopted the notion of using a tag (unique identifier) to identify indexes that have been retrieved to allow us to quickly access the indexes. As a result, anytime an index is retrieved, an index tag is assigned to it. For each index that is retrieved by the system, the tag must be readily available in order to manipulate the index. This run-time variable is designated by **Index Tag**.

A pointer into the index file must also be readily available to the system if the index is to be accessed at all. Clearly, if the index weren't going to be accessed at all, there would be little reason for the user to retrieve it. Therefore, a file descriptor for each index must also be kept as run-time information while the system is being used. This run-time variable is designated by **F_ptr**.

An index can be retrieved for either modification or read only. There are two pieces of run-time information that need to be kept related to the retrieval mode of indexes. The first is the actual retrieval mode. The system needs to know whether an index has been retrieved for modification or read only in order to prevent the user from trying to modify an index that was retrieved for read only. This is especially crucial in a mult-iuser environment, when more than one user may want to access the same index. This run-time variable is designated by **Mode**. Secondly, the system needs to keep track of whether the index has actually been modified (in the case of retrieval for modification). This information is used in the "save index" command. An index that has been retrieved for modification, but not actually modified does not need to be saved even if the user issues the save index command. Having this information available permits the system to detect these occurrences and not waste its time saving an index that has not actually changed. This run-time variable is designated by **Dirty**. **Dirty** is set to TRUE if the index has been modified, but not saved. **Dirty** is set to FALSE if the index has not been modified since the last time it was saved.

Finally, the system needs to know which indexes that are currently in the system have been created, but not saved. The reason for this is as follows. We cannot guarantee that a newly created index will be small enough to be completely contained in main memory. Therefore, when the user creates a new index, all persistent information is entered into the system catalog and the index files are created. The system needs to be able to distinguish these "created but not saved" indexes from those that either have been recently created but saved, or those that were retrieved. This distinction is necessary because if the user quits the system without saving these indexes, the system needs to know that they are to be deleted. This run-time variable is designated by **Saved**. **Saved** is set to TRUE if the index was retrieved during this user session (i.e. created sometime in the past) or if the index was created during this user session and has already been saved. **Saved** is set to FALSE if the index was created during this user session but has not yet been saved.

The remaining run-time information that needs to be available is the information found in the system catalog. Therefore, a pointer to the system catalog information is also needed at run-time. This run-time variable is designated by **SC_info**. Figure 6 shows the information that UIS needs to manage and manipulate indexes correctly.

### 2.2.2. Run-Time Information — Indexsets

As defined in Section 2, an indexset is a catalogued group of indexes and sets. Therefore, when an index or set is to be retrieved from an indexset, its system catalog information is found in the catalog components of the indexset (refer to Figure 1). At execution time, the system needs to maintain file descriptors to the catalog components of the indexset in order to be able to retrieve indexes and sets. These run-time variables are desig-

| Index Tag | Saved? | Mode | Dirty? | SC_info | F_ptr |
|---|---|---|---|---|---|
| *i1* | *FALSE* | *MODIFY* | *TRUE* | ** | *3* |
| *i2* | *TRUE* | *READ_ONLY* | *FALSE* | ** | *4* |

**Figure 6** - Run-Time Catalog Information for Indexes.

nated by **Fd_I_cat**, **Fd_I_attr_cat**, and **Fd_S_cat**. They correspond to the index catalog, the index attribute catalog and the set catalog components of the indexset, respectively.

If multiple indexes or sets are retrieved from a single indexset, we need to be very careful in making sure that only one set of catalog file descriptors are used for that indexset. If every retrieved index and set has its own file descriptor information for the indexsets catalog, then it would be very easy for the system to encounter read/write conflicts in the indexsets catalog components. Therefore, we need to have a way to maintain a single copy of the indexset information, and still know exactly how many indexes and sets from that indexset are currently retrieved. This suggests a need for run-time variables to count the number of retrieved indexes and sets for each indexset. This has two advantages. First, it prevents having multiple file descriptors to the indexset catalog components and prevents read/write conflicts. Second, it allows us to have the indexset retrieved for as small an amount of time as necessary. By keeping track of how many indexes and sets are currently retrieved, the system is able to return the indexset as soon as those numbers are zero. The run-time variables that designate these counts are **I_count** for indexes, and **S_count** for sets.

The remaining run-time information that needs to be available is the information found in the system catalog. Therefore, pointers to the system catalog information are also needed at run-time. These run-time variables are designated by **SC_info**, **Databook**, and **Indexspace**, which point to the different system catalog entries for the indexset. Figure 7 shows the information that UIS needs to manage and manipulate indexsets correctly.

### 2.2.3. Run-Time Information — Indexkits

There is no run-time information needed for indexkits. Because an indexkit is nothing more than a collection of system catalog information, all commands involving indexkits update only this system catalog

| Fd_I_cat | Fd_I_attr_cat | I_count | Fd_S_cat | S_count | SC_info | Databook | Indexspace |
|---|---|---|---|---|---|---|---|
| *3* | *4* | *2* | *5* | *0* | ** | | ## |
| *6* | *7* | *1* | *8* | *1* | ** | | ## |

**Figure 7** - Run-Time Catalog Information for Indexsets.

information. As a result, the catalog is only accessed at the exact moment a request is made. There is no notion of **retrieving** an indexkit, and at some later time making some modification to it.

# Part II: The Universal Index System Low-Level Design Document

## DESCRIPTION

User commands were designed at the same time as the system and run-time information was determined. The designs for indexes and sets were done first, since they are the fundamental objects managed by UIS. After those designs were almost complete, the indexkit and indexset commands were designed. This allowed us to isolate the differences between the objects at an early stage, and also allowed us to use our complete understanding of the index and set routines when trying to create an integrated design of the indexkit and indexset routines.

A template was used while creating the designs to facilitate a complete design. Each design contains the following sections: *System Requirements, System Architecture, System Data Structures, System Data Flow, System Control Flow, Design Rationale, Test Plan and Issues.* The System Requirements section describes the functionality of the command. It is taken from the requirements document. The System Architecture section presents an algorithm in pseudo-code describing what the system needs to do to execute the command. The System Data Structures section lists the input and output arguments needed to execute the command. Any error messages that are returned are also included in this section. The System Data Flow section provides a description of how the data flows among the different parts of the algorithm. The System Control Flow section provides a description of how execution control flows among the different parts of the algorithm.

The Design Rationale section gives a detailed explanation of the algorithm, and if necessary justifies why certain things are done, or why certain things need to be performed in a specific order. The Test Plan section suggests what type of tests should be run to properly and completely test the command and suggests some robustness tests. The Issues section discusses any side-effects of the routine, any hardware or software requirements for the execution of the routine, and provides explanations for any unclear information presented in the previous sections. The Issues section is also used to present unanswered questions about the design or the interaction of this routine with others.

The low-level design of the index, indexset and indexkit commands follows.

**Routine Name**: Create Index ( I_create() )

**Routine Number**: 3.1.2.2

## 1. System Requirements

Create index allows the user to create an index object. The user must specify the type of the index, the storage format, and the attributes and their type that make up the index. Formats include btrees, rtrees, ascii, virtual, etc. The type is expressed in the form (m,n) where m is the multiplicity of the item tuples, and n is the number of item tuples in the table.

## 2. System Architecture

parse
validate
create_index_files
allocate_WA
assign_tag
RTIC_insert
SIC_insert

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| index_name | string | < the name for the index to create > |
| indexspace | string | < indexspace to place index created > |
| Optional: | | |
| indexset | string | < indexset to place index created > |
| type | TYPE | < dimension of index > |
| format | string | < storage format: btree, rtree ... > |
| table | array of ATTR_DESC | < (attr_name,attr_type,attr_len) pairs for each attribute that makes up the index > |

### 2. Output

| | | |
|---|---|---|
| tag | TAG | < tag of index created, NULL if error > |

## 4. System Data Flow

parse --> validate
validate --> create_index_files
allocate_WA
assign_tag
RTIC_insert
SIC_insert
create_index_files --> RTIC_insert
SIC_insert
assign_tag --> RTIC_insert

SIC_insert

## 5. System Control Flow

I_create <-- parse
validate
assign_tag
create_index_files
RTIC_insert
SIC_insert

## 6. Design Rationale

This routine will create an empty index. First it will check that the index to create does not already exists. It will also check that the type and format arguments are valid. It will also check that the types of the attributes are valid attribute types. If none of these checks produces an error, then the index will be created, and a catalog entry will be created for it. A logical tag will be assigned to the newly created index. The catalog value for I_saved will be FALSE (meaning that this index has not been saved since creation time), index_mode will be 'm' (for modify), and for dirty will be 'true', so that the index will be saved as empty if no insertions are performed.

## 7. Test Plan

Test all supported formats, all supported types, creation of pre-existing index, insufficient arguments, and, of course, successful creation of an index. Also test invalid formats to make sure error is caught.

## 8. Issues

**Routine Name:** Drop Index ( I_drop() )

**Routine Number:** 3.1.8.1

## 1. System Requirements

The 'drop index' command allows the user to delete (or destroy) an index.

## 2. System Architecture

parse
validate
remove_files
SIKC_update(indexkit information update)
SIC_delete

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| index_name | string | < name of the index to delete > |
| indexset_name | string | < name of the indexset containing index > |

### 2. Output

| | |
|---|---|
| IME_OK | success |
| IME_FAILURE | general error |
| IME_DNE | index to drop does not exist |
| IME_BAD_MODE | index not retrieved for modification |

## 4. System Data Flow

| | | |
|---|---|---|
| parse | --> | validate |
| validate | --> | remove_files |
| | | SIKC_update |
| | | SIC_delete |

## 5. System Control Flow

| | |
|---|---|
| I_drop <-- | parse |
| | validate |
| | remove_files |
| | SIKC_update |
| | SIC_delete |

## 6. Design Rationale

This routine will delete an index from an indexset. First, it will check that the index to drop actually exists. If it does, then it will be deleted, and the catalogs will be updated. If the index has an indexkit associated with it, an indexkit call will be made to

update the indexkit catalog entry.

## 7. Test Plan

Test all error codes.

## 8. Issues

**Routine Name:** Insert Index ( I_Insert() )

**Routine Number:** 3.1.3.1.2

## 1. System Requirements

The 'insert into index' command allows one to modify an index by inserting a new row.

## 2. System Architecture

parse
validate

I_retrieve

for each tuple in the list
  IC_insert(attr_name attr_value pairs)

I_save

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| index_name | string | < name of Index to be updated > |
| indexset_name | string | < indexset name of Index to be updated > |
| attr_vals | array of ATTR_PAIR | < (attribute name, value) pairs for index entry to be inserted > |

### 2. Output

| | |
|---|---|
| IME_OK | I_insert successful |
| IME_FAILURE | general failure |
| IME_NONUNIQUE | non-unique indexset/name combination |
| IME_BAD_VALUE | bad pointer/attribute value |
| IME_DNE | index DNE |

## 4. System Data Flow

| | |
|---|---|
| parse --> | validate |
| validate --> | IC_insert |
| | I_Save |

## 5. System Control Flow

| | |
|---|---|
| I_Insert <-- | parse |
| | validate |
| | IC_insert |
| | I_Save |

## 6. Design Rationale

Tuple is inserted and Index is saved.

## 7. Test Plan

Test Cases should check for correct handling of invalid index names, bad pointer values, and attribute values that are not part of the index. Also check for attempt to insert incomplete tuples (containing NULLs).

## 8. Issues

**Routine Name:** Update Index ( I_Update() )

**Routine Number:** 3.1.3.1.1

## 1. System Requirements

The 'update index' command updates the specified index using the given attribute and pointer values. The user must specify a valid index name and indexset name, attribute value(s), and/or pointer value(s), using the correct syntax to indicate whether the pointer or the attribute is being altered.

## 2. System Architecture

parse
validate

IB_build

I_first
I_fetch
alter_tuple
IC_delete
IC_insert

(repeat as necessary the 6 following commands)
I_next
I_fetch
alter_tuple
IC_delete
IC_insert
I_save

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| index_name | string | < name of Index to be updated > |
| indexset_name | string | < indexset name of Index to be updated > |
| set_vals | array of ATTR_PAIR | < array of (attribute name, value) pairs to be passed to IC_Update > |
| bool_val | string | < condition to be used to build a boolean to traverse the index and select tuples to be modified > |

### 2. Output

| | |
|---|---|
| IME_OK | I_update successful |
| IME_FAILURE | general failure |
| IME_NONUNIQUE | non-unique indexset/name combination |
| IME_BAD_VALUE | bad attribute value |
| IME_DNE | index DNE |

## 4. System Data Flow

| | |
|---|---|
| parse --> | validate |
| validate --> | IB_build |
| | I_first |
| | I_fetch |
| | alter_tuple |
| | IC_delete |
| | IC_inset |
| | I_next |
| | I_save |

## 5. System Control Flow

| | |
|---|---|
| I_update<-- | parse |
| | validate |
| | IB_build |
| | I_first |
| | I_fetch |
| | alter_tuple |
| | IC_delete |
| | IC_insert |
| | I_next |
| | I_save |

## 6. Design Rationale

Get the index to be updated. Fetch tuples by traversing the index using the boolean condition given in the update command (bool_val). Update each qualifying tuple by setting the attributes/pointer as specified in the update command (set_vals). When no more tuples qualify, save the changes.

## 7. Test Plan

Test Cases should check for correct handling of invalid index names, bad pointer values, bad data values.

## 8. Issues

**Routine Name:** Move Index ( I_move() )

**Routine Number:** 3.1.5.1

## 1. System Requirements

The move index command allows the user to move an index from one location to another. ( potentially renaming the index in the process).

## 2. System Architecture

parse
validate
check_uniqueness
move_index

RTIC_retrieve < update both indexset catalogs >
RTIC_delete
RTIC_insert
SIC_retrieve < update both indexset catalogs >
SIC_delete
SIC_insert

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| index_name | string | < name of the index to move > |
| old_indexset | string | < current location of index > |
| new_indexset | string | < place to move and name to give index > |

### 2. Output

| | |
|---|---|
| IME_OK | success |
| IME_FAILURE | general error |
| IME_DNE | index to move non-existent |
| IME_NONUNIQUE | index to move will not be unique in new location |

## 4. System Data Flow

| | | |
|---|---|---|
| parse | --> | validate |
| validate | --> | check_uniqueness |
| | | move_index |
| move_index | --> | RTIC_retrieve |
| | | RTIC_delete |
| | | RTIC_insert |
| | | SIC_retrieve |
| | | SIC_delete |
| | | SIC_insert |

## 5. System Control Flow

| | |
|---|---|
| I_move <-- | parse |
| | validate |
| | check_uniqueness |
| | move_index |
| | RTIC_retrieve |
| | RTIC_delete |
| | RTIC_insert |
| | SIC_retrieve |
| | SIC_delete |
| | SIC_insert |

## 6. Design Rationale

This routine will move an index from one indexset to another. First it will check that the index to move currently exists and will remain unique in its new indexset. If so, it will move the index files from one indexset to another, updating the catalogs of both indexsets to reflect the change.

## 7. Test Plan

Test all error codes.

## 8. Issues

**Routine Name:** Delete from Index (I_Delete() )

**Routine Number:** 3.1.3.1.3

## 1. System Requirements

The 'delete from index' command allows one to modify an index by removing one index tuple or all the tuples with a given attribute value.

## 2. System Architecture

parse
validate

IB_build

I_first

IC_delete

(repeat as necessary the 3 following commands)
I_next
IC_delete

I_save

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| index_name | string | < name of Index to be modified > |
| indexset_name | string | < indexset name of Index to be modified > |
| bool_val | string | < boolean condition for traversing the index and selecting tuples to be deleted > |

### 2. Output

| | |
|---|---|
| IME_OK | I_delete successful |
| IME_NONUNIQUE | non-unique indexset/name combination |
| IME_BAD_VALUE | bad attribute/pointer value |
| IME_DNE | index DNE |

## 4. System Data Flow

| | | |
|---|---|---|
| parse | --> | validate |
| validate | --> | IB_build |
| | | I_first |
| | | IC_delete |
| | | I_next |
| | | IC_delete |
| | | I_save |

## 5. System Control Flow

| | | |
|---|---|---|
| I_delete | <-- | parse |
| | | validate |
| | | IB_build |
| | | I_first |
| | | IC_delete |
| | | I_next |
| | | IC_delete |
| | | I_save |

## 6. Design Rationale

Open the index. Position the pointer to point to a tuple, using the boolean built with attr_vals. Delete that tuple, move on to the next one.
When no more tuples qualify, save the index.

## 7. Test Plan

Test Cases should check for correct handling of invalid index names, bad pointer values, and bad data values.

## 8. Issues

**Routine Name:** Copy Index ( I_copy() )

**Routine Number:** 3.1.4.1.1

## 1. System Requirements

The copy index command allows the user to make identical copies of indexes.

## 2. System Architecture

[ B = copy(A) ]

parse
validate

I_retrieve(A)
I_create (B)

for each tuple in A
    insert_tuple into B
end loop

I_save (B)
I_return(A)
I_return(B)

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| index1 | string | < name for index to be copied from > |
| indexset1 | string | < indexset containing index1 > |
| index2 | string | < name of the index to copy into > |
| indexset2 | string | < indexset to contain index2 > |
| Optional: | | |
| format | string | < storage representation of index > |

### 2. Output

| | |
|---|---|
| IME_OK | success |
| IME_FAILURE | general error |
| IME_DNE | index to copy does not exist |
| IME_NONUNIQUE | new index name already exists |

## 4. System Data Flow

| | |
|---|---|
| parse --> | validate |
| validate --> | I_retrieve |
| | I_create |
| | copy_loop |
| | I_save |
| | I_return |

I_return

## 5. System Control Flow

I_copy <-- parse
validate
I_retrieve
I_create
copy_loop
I_save
I_return
I_return

## 6. Design Rationale

This routine will make a duplicate copy of the specified index (with potentially a different format). First it will check that the index to copy exists, and the resulting index will be unique. If no errors are produced from these checks, then the new index will be created with the same parameters as the index to be copied. The index name and format are taken from the argument list. Specifying the format is optional. If no format is specified, it is taken from the catalog entry of the index to copy. The index is copied, and the catalog is updated. The new index is saved and both indexes are returned.

## 7. Test Plan

Test all error codes.

## 8. Issues

**Routine Name:** Intersect Index ( I_intersect() )

**Routine Number:** 3.1.4.1.4

## 1. System Requirements

The 'intersect index' command allows the user to intersect indexes.

## 2. System Architecture

[ C = A - B ]

parse
validate

I_retrieve(A)
I_retrieve(B)
test_compatibility(A,B)
I_create (C)

for every tuple in A
    if there exists an equivalent tuple in B then
        insert_tuple(C)
end loop

I_save (C)
I_return(A)
I_return(B)
I_return(C)

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| index1 | string | < name of one index to intersect > |
| indexset1 | string | < indexset containing index1 > |
| index2 | string | < name of the other index to intersect > |
| indexset2 | string | < indexset containing index2 > |
| new_index | string | < name of the new index to create > |
| indexset3 | string | < indexset containing new_index > |
| Optional: | | |
| new_index_format | string | < see Create Index 3.3.2 > |

### 2. Output

| | |
|---|---|
| IME_OK | success |
| IME_FAILURE | general error |
| IME_DNE | an index to intersect does not exist |
| IME_NONUNIQUE | index to create as result already exists |
| IME_INCOMPATIBLE | indexes to intersect are of incompatible types |

## 4. System Data Flow

```
parse    -->   validate
validate -->   I_retrieve
               I_retrieve
               test_compatibility
               I_create
               intersection_loop
               I_save
               I_return
               I_return
               I_return
```

## 5. System Control Flow

```
I_intersect <-- parse
                validate
                I_retrieve
                test_compatibility
                I_create
                intersection_loop
                I_save
                I_return
                I_return
                I_return
```

## 6. Design Rationale

This routine will compute the intersection of two indexes. First it will check that the two indexes to intersect actually exist and are of compatible types. It will also check that the resulting index does not already exist. If none of these checks produces an error, then it will create the new index, compute the intersection, and save the newly created index. The arguments passed to I_create will be obtained from the catalog entry of the first index to be intersected. The name of the index and the format of the index are provided by the argument list. Specifying the format is optional. If it is not specified, it will be taken from the first index to be intersected.

## 7. Test Plan

Test all error codes.

## 8. Issues

**Routine Name:** Subset Index ( I_subset() )

**Routine Number:** 3.1.4.1.2

## 1. System Requirements

The 'subset index' command allows one to make an index from a subset of another index.

## 2. System Architecture

[ B = subset(A) ]

parse
validate

I_retrieve(A)

IS_syntax(select)
IB_build(boolean)

I_create(B)

get first record in index
test record with boolean
fetch record from index
insert into index using select

(repeat as necessary the 4 following commands)
get next record in index
test record with boolean
fetch record from index using select
insert into index

I_save(B)
I_return(A)
I_return(B)

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| index1 | string | < name of index to be created > |
| indexset1 | string | < indexset containing index1 > |
| index2 | string | < name of index to be subsetted > |
| indexset2 | string | < indexset to contain index2 > |
| select | string | < attribute names of indexed attrs to be subsetted > |
| bool_str | string | < boolean condition used to select tuples to go into new index > |
| format | string | < indicates r-tree, b-tree, or ascii > |

### 2. Output

| | |
|---|---|
| IME_OK | successful creation of subset index |
| IME_FAILURE | general failure |
| IME_NONUNIQUE | non-unique indexset/name combination |
| IME_DNE | index to subset from does not exist |

4. **System Data Flow**

parse --> validate
validate --> I_retrieve
IS_syntax
IB_build
I_create
get_first_rec
fetch_rec_w_select
insert_rec
get_next_rec
fetch_rec_w_select
insert_rec
I_save
I_return
I_return

5. **System Control Flow**

I_subset <-- parse
validate
I_retrieve
IS_syntax
IB_build
I_create
get_first_rec
fetch_rec_w_select
insert_rec
get_next_rec
I_save
I_return
I_return

6. **Design Rationale**

Check that index to subset exists, build select, build boolean, check that the index to create with result does not already exist. Create the new index. Navigate the index. Insert the selected parts of the retrieved tuples into the newly created index. Save the new index. Return the two indexes.

7. **Test Plan**

Test Cases should check for correct handling of invalid index names, bad format values.

8. **Issues**

**Routine Name:** Subtract Index ( I_subtract() )

**Routine Number:** 3.1.4.1.5

## 1. System Requirements

The subtract index command allows the user to subtract the different parts of indexes.

## 2. System Architecture

[ C = A - B ]

parse
validate

I_retrieve(A)
I_retrieve(B)
test_compatibility(A,B)
I_create (C)

for every tuple in A
    if there does not exist an equivalent tuple in B then
        insert_tuple(C)
end loop

I_save (C)
I_return(A)
I_return(B)
I_return(C)

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| index1 | string | < name of one index to subtract > |
| indexset1 | string | < indexset of index1 > |
| index2 | string | < name of the other index to subtract > |
| indexset2 | string | < indexset of index2 > |
| index3 | string | < name of the new index to create > |
| indexset3 | string | < indexset of new index > |
| Optional: | | |
| new_index_format | string | < see Create Index 3.3.2 > |

### 2. Output

| | |
|---|---|
| IME_OK | success |
| IME_FAILURE | general error |
| IME_DNE | an index to subtract does not exist |
| IME_INCOMPATIBLE | indexes to subtract are of incompatible types |
| IME_NON_UNIQUE | index to create as result already exists |

## 4. System Data Flow

```
parse    -->  validate
validate -->  I_retrieve
              I_retrieve
              test_compatibility
              I_create
              subtraction_loop
              I_save
              I_return
              I_return
              I_return
```

## 5. System Control Flow

```
I_subtract <-- parse
               validate
               I_retrieve
               I_retrieve
               test_compatibility
               I_create
               subtraction_loop
               I_save
               I_return
               I_return
               I_return
```

## 6. Design Rationale

This routine will compute the subtraction of two indexes. First it will check that the two indexes to subtract actually exist and are of compatible types. It will also check that the resulting index does not already exist. If none of these checks produces an error, then it will create the new index, compute the subtraction, and save the newly created index. The arguments passed to I_create will be obtained from the catalog entry of the first index to be subtracted. The name of the index and the format of the index are provided by the argument list. Specifying the format is optional. If it is not specified, it will be taken from the first index to be subtracted.

## 7. Test Plan

Test all error codes.

## 8. Issues

**Routine Name:** Union Indexes ( I_union() )

**Routine Number:** 3.1.4.1.3

## 1. System Requirements

The 'union index' command allows one to create an index that is the union of two other indexes.

## 2. System Architecture

[ C = A union B ]

parse
validate

I_retrieve(A)
I_retrieve(B)
test_compatibility(A,B)
I_create(C)

(for each index being unioned)
get first record in index
fetch record from index
insert into index

(repeat as necessary the 3 following commands)
next record in index
fetch record from index
insert into index

(end repeat for each index)

I_save(C)
I_return(A)
I_return(B)
I_return(C)

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| index1 | string | < name of index to be unioned > |
| indexset1 | string | < indexset containing index1 > |
| index2 | string | < name of index to be unioned > |
| indexset2 | string | < indexset containing index2 > |
| new_index | string | < name of index to be created > |
| indexset3 | string | < indexset to contain new index > |
| format | string | < indicates r-tree, b-tree, or ascii > |

### 2. Output

| | |
|---|---|
| IME_OK | successful union index |

| | |
|---|---|
| IME_FAILURE | general failure |
| IME_DNE | failure: index does not exist |
| IME_NONUNIQUE | non-unique indexset/name combination |
| IME_INCOMPATIBLE | indexes to union are incompatible |

## 4. System Data Flow

parse --> validate
validate --> I_retrieve
I_retrieve
test_compatibility
I_create
union_loop
I_save
I_return
I_return
I_return

## 5. System Control Flow

I_union <-- parse
validate
I_retrieve
I_retrieve
test_compatibility
I_create
union_loop
I_save
I_return
I_return
I_return

## 6. Design Rationale

Check that indexes to union exist and are compatible, and check that the index to create as result does not already exist.
Create the new index. Navigate the indexes being unioned, retrieving tuples and inserting them into the newly created index. Save the newly created index, and return all indexes retrieved and created.

## 7. Test Plan

Test Cases should check for correct handling of invalid index names, bad format values.

## 8. Issues

Boolean is TRUE and Select is *.

**Routine Name:** Find Term in Index ( I_find_term() )

**Routine Number:** 4.1.1.3

## 1. System Requirements

The 'find term in index' command will enable the user to obtain from the current index component a list of terms alphabetically surrounding a submitted term in the logical text file.

In the Menu access, the default editor is automatically invoked to read the logical text file. In the Host Language Interface, the result can also be in the logical text string.

## 2. System Architecture

process form
parse
validate

IB_build

get first record in index
test record with boolean
fetch record from index
add to term list

(repeat as necessary the 3 following commands)
get next in index
test record with boolean
fetch record from index
add to term list

Return Term List

## 3. System Data Structures

1. **Input** (input is gotten by Process_form and returned in a string)

| | | |
|---|---|---|
| term_string | string | < string form "attr_name attr_value" > |

2. **Output**

| | | |
|---|---|---|
| term_list | array of strings | < each string is a found term. NULL value indicates error or no terms found > |

## 4. System Data Flow

| | | |
|---|---|---|
| form_new | --> | parse |
| parse | --> | validate |
| validate | --> | IB_build |
| | | get_first_rec |

fetch_rec
add_to_term_list
get_next_rec
fetch_rec
add_to_term_list
return_term_list

## 5. System Control Flow

I_Find_Term<-- form_new
parse
validate
IB_build
get_first_rec
fetch_rec
add_to_term_list
get_next_rec
return_term_list

## 6. Design Rationale

A boolean is built using 'term' and some pre-set range decided by the system to be an acceptable alphabetical range for single term queries. The index is navigated and terms are added to the term-list as they are fetched. The term-list is returned when the entire index has been searched.

## 7. Test Plan

Test Cases should check for correct handling of invalid attribute names and bad term values.

## 8. Issues

The pre-defined ranges are set in the system.

**Routine Name**: Build Set from Index with Term ( I_build_set_term() )

**Routine Number**: 4.1.1.4.1

## 1. System Requirements

The 'build set with term' command will enable the user to build a set of Entrids which are associated with a submitted term in the current index. The system will assign a tag to the new set and print in the logical text file the set tag, set description, and set size.

In the Menu access, the logical text file is automatically displayed. In the Host Language Interface access, the result can also be in the logical text string.

## 2. System Architecture

parse
validate

IB_build

S_build_empty
get first record in index
fetch record from index using select
insert into set

(repeat as necessary the 3 following commands)
next in index
fetch record from index using select
insert into set

return set description

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| set_name | string | < name of the set to be built > |
| attr_name | string | < name of the attribute for the term > |
| term | string | < term to be searched for > |
| set_attr | string | < attribute to put build set from > |

### 2. Output

| | | |
|---|---|---|
| set_desc | SET_INFO | < string containing set tag, set description, and set size. NULL value indicates error > |

## 4. System Data Flow

parse --> validate
validate --> S_build_empty

IB_build
get_first_rec
fetch_rec
insert_into_set
get_next_rec
fetch_rec
insert_into_set
return_set_desc

## 5. System Control Flow

| | |
|---|---|
| I_build_set_w_term <-- | parse |
| | validate |
| | S_build_empty |
| | IB_build |
| | get_first_rec |
| | test_w_boolean |
| | fetch_rec |
| | insert_into_set |
| | get_next_rec |
| | return_set_desc |

## 6. Design Rationale

A boolean is built using 'term'. The index is navigated and pointer values are added to the set as they are fetched.
The set description is returned when the entire index has been searched.

## 7. Test Plan

Test Cases should check for correct handling of invalid attribute names and bad term values.

## 8. Issues

**Routine Name:** Build Set from Index with List ( I_build_set_list() )

**Routine Number:** 4.1.1.4.2

## 1. System Requirements

The 'build set with list' command will enable the user to build a set of Entrids which are associated with a list of terms contained in a submitted file in the current index. The system will assign a tag to the new set and print in the logical text file the set tag, set description, and set size.

In the Menu access, the logical text file is automatically displayed. In the Host Language Interface access, the result can also be in the logical text string.

## 2. System Architecture

parse
validate

IB_build
S_build_empty

get first record in index
test record with boolean
fetch record from index
insert into set

(repeat as necessary the 3 following commands)
next in index
test record with boolean
fetch record from index
insert into set

return set description

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| set_name | string | < name of the set to be built > |
| attr_name | string | < name of the attribute for the terms > |
| term_list | array of strings | < terms to be searched for > |
| set_attr | string | < attribute to project into set > |

### 2. Output

| | | |
|---|---|---|
| set_desc | SET_INFO | < contains set tag, set description, and set size. NULL value indicates error > |

## 4. System Data Flow

| | |
|---|---|
| parse --> | validate |
| validate --> | S_build_empty |
| | IB_build |
| | get_first_rec |
| | fetch_record |
| | insert_into_set |
| | get_next_rec |
| | fetch_rec |
| | insert_into_set |
| | return_set_desc |

## 5. System Control Flow

| | |
|---|---|
| I_build_set_list <-- | parse |
| | validate |
| | S_build_empty |
| | IB_build |
| | get_first_rec |
| | fetch_record |
| | insert_into_set |
| | get_next_rec |
| | return_set_desc |

## 6. Design Rationale

A boolean is built using the terms in 'term_list'. The index is navigated and pointer values are added to the set as they are fetched. The set description is returned when the entire index has been searched for every term in the list.

## 7. Test Plan

Test Cases should check for correct handling of invalid attribute names and bad term values.

## 8. Issues

**Routine Name**: Build Set from Index with Range ( I_build_set_range() )

**Routine Number**: 4.1.1.4.3

## 1. System Requirements

The 'build set with range' command will enable the user to build a set of Entrids which are associated with a boolean expression in the current index. The system will assign a tag to the new set and print in the logical text file the set tag, set description, and set size.

In the Menu access, the logical text file is automatically displayed. In Host Language Interface access, the result can also be int the logical text string.

## 2. System Architecture

parse
validate

IB_build

S_build_empty

get first record in index
test record with boolean
fetch record from index
insert into set

(repeat as necessary the 4 following commands)
get next record in index
test record with boolean
fetch record from index
insert into set

return set description

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| set_name | string | < name of the set to be built > |
| attr_name | string | < name of attribute having values > |
| term_list | arry of strings | < values to create range from > |
| set_attr | string | < attribute to project into set > |

E.g. If term_list = { 1, 12, 18, 22, 29, 33, NULL }
then that means get all tuples from the current index
where 1 < attr_name < 12 OR
18 < attr_name < 22 OR
29 < attr_name < 33

### 2. Output

| | | |
|---|---|---|
| set_desc | SET_INFO | < entries contain set tag, set description, and set size. NULL indicates error > |

## 4. System Data Flow

| | |
|---|---|
| parse --> | validate |
| validate --> | S_build_empty |
| | IB_build |
| | get_first_rec |
| | fetch_rec |
| | insert_into_set |
| | get_next_rec |
| | fetch_rec |
| | insert_into_set |
| | return_set_desc |

## 5. System Control Flow

| | |
|---|---|
| I_build_set_w_range<-- | parse |
| | validate |
| | S_build_empty |
| | IB_build |
| | get_first_rec |
| | fetch_rec |
| | insert_into_set |
| | get_next_rec |
| | insert_into_set |
| | return_set_desc |

## 6. Design Rationale

A boolean is built using the information in term_list to build the range as described above. The index is navigated and pointer values are added to the set as they are fetched. The set description is returned when the entire index has been searched.

## 7. Test Plan

Test Cases should check for correct handling of invalid attribute names and bad term values.

## 8. Issues

**Routine Name**: Retrieve Index ( I_retrieve() )

**Routine Number**: 4.1.1.1.1

## 1. System Requirements

The 'retrieve index' command enables the user to retrieve or open an index component. The retrieval can be for 'read-only' or 'modify' mode. The system will automatically assign the retrieved component to a logical tag. If the component does not exist, then an error message appears in the logical error file.

## 2. System Architecture

parse
validate

SIC_retrieve
RTIC_insert

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| index_name | string | < name of index to be retrieved > |
| indexset_name | string | < indexset name of index to be retrieved > |
| mode | integer | < signifies read-only or modify mode > |

### 2. Output

| | |
|---|---|
| IME_OK | successful retrieval of index |
| IME_FAILURE | general failure |
| IME_DNE | index does not exist |

## 4. System Data Flow

| | |
|---|---|
| parse --> | validate |
| validate --> | SIC_retrieve |
| | RTIC_insert |

## 5. System Control Flow

| | |
|---|---|
| I_retrieve <-- | parse |
| | validate |
| | SIC_retrieve |
| | RTIC_insert |

## 6. Design Rationale

First check that the index is not already retrieved, and if so, return an error. Otherwise, retrieve the system catalog information, assign it a tag, and update the RT catalogs to

reflect this.

## 7. Test Plan

Test all error codes, plus the retrieval of an already retrieved index.

## 8. Issues

Should you be allowed to retrieve the same index more than once?
-->NO

**Routine Name**: Pick Index ( I_pick() )

**Routine Number**: 4.1.1.1.3

## 1. System Requirements

The 'pick index' command will enable the user to make as 'current' one of the retrieved index components. If the component does not exist, then an error message appears in the logical error file.

## 2. System Architecture

parse
validate

RTIC_retrieve
I_bind_retrieve
CWA_I_update

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| tag | TAG | < identifier for retrieved index > |

### 2. Output

| | |
|---|---|
| IME_OK | successful picking of index |
| IME_BAD_TAG | bad tag |

## 4. System Data Flow

| | |
|---|---|
| parse --> | validate |
| validate --> | RTIC_retrieve |
| | CWA_I_update |

## 5. System Control Flow

| | |
|---|---|
| I_pick <-- | parse |
| | validate |
| | RTIC_retrieve |
| | CWA_I_update |

## 6. Design Rationale

Search the RTC for the tag of the index to be made current. When a match is found, update the CWA to point to the RTC entry for that index. Also, update the pointer to current binding for the newly picked index.

## 7. Test Plan

Test for invalid tag values and tags that don't belong to an index

component.

## 8. Issues

The current index boolean and current index select are automatically updated to the default boolean (TRUE) and select (*), to gurantee that at all times the boolean and select are compatible with the

**Routine Name**: Save Index ( I_save() )

**Routine Number**: 4.1.1.1.4

## 1. System Requirements

The 'save index' command enables the user to save the 'current' retrieved index component. If the component was not retrieved in 'modify' mode, then an error message appears in the logical error file.

## 2. System Architecture

parse
validate

flush buffers to disk

RTIC_update

## 3. System Data Structures

### 1. Input

### 2. Output

| | |
|---|---|
| IME_OK | successful save |
| IME_FAILURE | general failure |
| IME_BAD_MODE | wrong mode on index |
| IME_BAD_TAG | invalid tag |

## 4. System Data Flow

| | |
|---|---|
| parse --> | validate |
| validate --> | flush_buffers_to_disk |
| | RTIC_update |

## 5. System Control Flow

| | |
|---|---|
| I_save <-- | parse |
| | validate |
| | flush_buffers_to_disk |
| | RTIC_update |

## 6. Design Rationale

After validating that the index is open for modify mode and has been modified since the last save, flush the buffers containing the updated copy of the index to disk. Update the RTC to reflect that the index has not been modified since the last save.

## 7. Test Plan

Test for bad mode.

**Routine Name:** Return Index ( I_return() )

**Routine Number:** 4.1.1.1.5

## 1. System Requirements

The 'return index' command will enable the user to return a retrieved index component. If the index was retrieved for modify mode and was not saved, the an error message appears in the logical error file.

## 2. System Architecture

parse
validate

RTIC_delete

CWA_I_update (if necessary)
I_bind_delete

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| tag | TAG | < tag for index to be returned > |

### 2. Output

| | |
|---|---|
| IME_OK | successful save |
| IME_FAILURE | general failure |
| IME_NOTSAVED | index modified but not saved |
| IME_BAD_TAG | bad tag value |

## 4. System Data Flow

| | |
|---|---|
| parse --> | validate |
| validate --> | RTIC_delete |
| | CWA_I_update |
| | I_bind_delete |

## 5. System Control Flow

| | |
|---|---|
| I_return <-- | parse |
| | validate |
| | RTIC_delete |
| | CWA_I_update |
| | I_bind_delete |

## 6. Design Rationale

First check to see if any modification has been done since the last save, and if none has, remove its entry from the RTC. If the index you are returning is the current index, then the CWA for indexes will

have to be updated. If the index has been modified, it will still be returned, but an error message is returned.

## 7. Test Plan

Test for bad tags and for an attempt to return a modified index before saving it.

## 8. Issues

Since minirel is being used as the underlying system, when an index that has been modified is returned, it is considered saved. In other words, the modifications become permanent.

**Routine Name:** List Indexes ( I_list() )

**Routine Number:** 4.1.1.1.2

1. **System Requirements**

   The 'list indexes' command will enable the user to obtain a list of the retrieved index components of the current book.

   In the Menu access, the default text editor is automatically invoked to read the logical text file. In Host Language Interface access, the result can also be in the logical text string.

2. **System Architecture**

   RTIC_traverse

3. **System Data Structures**

   1. **Input**

   2. **Output**

| | | |
|---|---|---|
| val_list | LIST | < a structure containing the number of indexes currently retrieved, and an array of tag, index-name pairs about each index > |

4. **System Data Flow**

5. **System Control Flow**

   I_list <-- RTIC_traverse

6. **Design Rationale**

   Search the run-time catalog for indexes, return the tag, name, and indexset name of each index entry found.

7. **Test Plan**

   Test for case where there are no retrieved indexes.

8. **Issues**

**Routine Name:** First in Index ( I_first() )

**Routine Number:** 4.1.1.5.1.1

## 1. System Requirements

The 'first in index' command will enable the user to position the curent row pointer to point to the first row of the current index component with respect to the current index boolean (see section 4.3.5.2). If no such row is found, then an error is indicated in the logical error file.

## 2. System Architecture

parse
validate

search with boolean
I_bind_info
CWA_I_update

## 3. System Data Structures

### 1. Input

### 2. Output

| | |
|---|---|
| IME_OK | successful search |
| IME_FAILURE | general failure |
| IME_NO_CURRENT | no current index |
| IME_NO_QUALIFY | no qualifying row |

## 4. System Data Flow

| | | |
|---|---|---|
| parse | --> | validate |
| validate | --> | search_w_boolean |
| | | I_bind_info |
| | | CWA_I_update |

## 5. System Control Flow

| | | |
|---|---|---|
| I_first | <-- | parse |
| | | validate |
| | | search_w_boolean |
| | | I_bind_info |
| | | CWA_I_update |

## 6. Design Rationale

This routine will search from the beginning of the index file, looking for the first tuple in the index to satisfy the current index boolean. I_current_row is updated. Binding to program variables is performed.

## 7. Test Plan

Test all error codes.

## 8. Issues

**Routine Name**: Next in Index ( I_next() )

**Routine Number**: 4.1.1.5.1.2

## 1. System Requirements

The 'next in index' command will enable the user to position the current row pointer in the CWA to point to the next row of the current index component with respect to the current index boolean (see section 4.3.5.2). If no such row is found, then an error is indicated in the logical error file.

## 2. System Architecture

parse
validate

relative search with boolean

I_bind_info
CWA_I_update(I_current_row)

## 3. System Data Structures

### 1. Input

### 2. Output

| | |
|---|---|
| IME_OK | successful search |
| IME_FAILURE | general failure |
| IME_NO_CURRENT | no current row |
| IME_NO_QUALIFY | no qualifying row |

## 4. System Data Flow

| | |
|---|---|
| parse --> | validate |
| validate --> | relative_search_w_boolean |
| | I_bind_info |
| | CWA_I_update |

## 5. System Control Flow

| | |
|---|---|
| I_next <-- | parse |
| | validate |
| | relative_search_w_boolean |
| | I_bind_info |
| | CWA_I_update |

## 6. Design Rationale

Relative_Search_w_Boolean will search from the value of I_current_row forward through the index, looking for the next tuple that satisfies the current index boolean. Binding to program variables is performed.

7. **Test Plan**

Test for case with no qualifying row.

8. **Issues**

**Routine Name:** Fetch from Index ( I_fetch() )

**Routine Number:** 4.1.1.5.1.4

## 1. System Requirements

The 'fetch from index' command will enable the user to obtain a row of the current index component with respect to the current index row address. The row is output to the logical text file. The amount of the row output is defined by the current index select (see section 4.3.5.3).

In the Menu access, the default text editor is automatically invoked to read the logical text file. In Host Language Interface access, the result can also be in the logical text file, logical text string, or assigned to program variables (see section 4.3.7).

## 2. System Architecture

parse
validate
return tuple

## 3. System Data Structures

### 1. Input

### 2. Output

| | | |
|---|---|---|
| tuple_val | TUPLE | < selected parts of the current index row will be null if there's an error > |

## 4. System Data Flow

parse --> validate
validate --> return_tuple

## 5. System Control Flow

I_fetch <-- parse
validate
return_tuple

## 6. Design Rationale

Return the tuple pointed to by the current row pointer in the current working area.

## 7. Test Plan

## 8. Issues

**Routine Name**: Last in Index ( I_last() )

**Routine Number**: 4.1.1.5.1.5

## 1. System Requirements

The 'last in index' command will enable the user to position the current row point of the CWA to point to the last row of the current index component with respect to the current index boolean (see section 4.3.5.2). If no such row is found, then an error is indicated in the logical error file.

## 2. System Architecture

parse
validate

backward search with boolean

I_bind_info
CWA_I_update

## 3. System Data Structures

### 1. Input

### 2. Output

| | |
|---|---|
| IME_OK | successful search |
| IME_FAILURE | general failure |
| IME_NO_CURRENT | no current index |
| IME_NO_QUALIFY | no qualifying row |

## 4. System Data Flow

| | |
|---|---|
| parse --> | validate |
| validate --> | backward_search_w_boolean |
| | I_bind_info |
| | CWA_I_update |

## 5. System Control Flow

| | |
|---|---|
| I_last <-- | parse |
| | validate |
| | backward_search_w_boolean |
| | I_bind_info |
| | CWA_I_update |

## 6. Design Rationale

This routine will search backward from the end of the index file looking for the first (remember searching backward) tuple that satisfies the current index boolean. The value of I_current_row is

updated. Binding to program variables is performed.

7. **Test Plan**

Test all error codes.

8. **Issues**

**Routine Name**: Previous in Index ( I_previous() )

**Routine Number**: 4.1.1.5.1.3

## 1. System Requirements

The 'previous in index' command will enable the user to position the current row pointer in the CWA to point to the previous row of the current index component with respect to the current index boolean (see section 4.3.5.2). If no such row is found, then an error is indicated in the logical error file.

## 2. System Architecture

parse
validate

relative backward search with boolean

I_bind_info
CWA_I_update

## 3. System Data Structures

### 1. Input

### 2. Output

| | |
|---|---|
| IME_OK | successful search |
| IME_FAILURE | general failure |
| IME_NO_CURRENT | no current row |
| IME_NO_QUALIFY | no qualifying row |

## 4. System Data Flow

parse --> validate
validate --> relative_backward_search_w_boolean
I_bind_info
CWA_I_update

## 5. System Control Flow

I_previous <-- parse
validate
relative_backward_search_w_boolean
I_bind_info
CWA_I_update

## 6. Design Rationale

Get the previous row wrt the current row and current boolean. Update the CWA current row pointer. Binding to program variables is performed.

## 7. Test Plan

Test all error codes.

## 8. Issues

**Routine Name:** Build Index Boolean ( IB_build() )

**Routine Number:** 4.1.1.5.2.1

## 1. System Requirements

The system will enable the user to build an index boolean for navigating through the index rows. The user receives in the logical text file an index boolean form for building such a boolean. The user completes the form and submits it. The system assigns the index boolean a tag. If an error in building the boolean is made, then an error is returned.

In Menu and Command access, the form is input/ouput through the logical text file. In the Menu access, the default text editor is automatically invoked to build the form. In Host Language Interface access, the form can be built through the logical text string.

## 2. System Architecture

form_new
parse
validate
assign_tag
RTI_BC_insert

## 3. System Data Structures

### 1. Input

### 2. Output

| | | |
|---|---|---|
| tag | TAG | the tag of the index boolean just created. NULL if error occurs. |

## 4. System Data Flow

| | | |
|---|---|---|
| form_new | --> | parse |
| parse | --> | validate |
| validate | --> | assign_tag |
| | | RTI_SC_insert |

## 5. System Control Flow

| | |
|---|---|
| IS_build <-- | form_new |
| | parse |
| | validate |
| | assign_tag |
| | RTI_SC_insert |

## 6. Design Rationale

This routine will build an index boolean and insert it into the

catalog. The user is given a form to complete defining the index boolean. This definition is then parsed into an internal format, checking that the expression is syntactically correct. A catalog entry is created and a tag is assigned to the boolean definition. There is no compatibility check done at this time. That is done at the time the user 'picks' the index Boolean as current.

## 7. Test Plan

## 8. Issues

This routine receives it's input from form_new that is spawned. The form_new will give this routine's input in string format.

| | | |
|---|---|---|
| Boolean_list | string | < boolean condition > |

It is up to us to parse the string into our internal format.

A valid boolean condition must follow the following rules:
Relational operators supported are <, >, =, <>, <=, >=.
Logical operators supported are AND, OR, NOT.
Attribute types supported are characters, integers, floats, and strings.
A simple clause is of the form:
[NOT] <attr_name> <rel_op> { <attr_name> | <constant> }
NOT is optional, and if specified implies the negation of the entire simple clause. The first operand MUST ALWAYS be an attribute name. The second operand can be either an attribute name or a constant.
Constants are of the following forms:

| | | |
|---|---|---|
| Ingeter constants | the number | e.g. 345 10 0 -3 |
| Float constants | the number | e.g. 3.1415 0.3 -3.553 |
| Character constants | the character enclosed in single quotes | e.g. 'a' 'x' '?' '@' '1' |
| String constants | the string enclosed in double quotes (note the quotes are NOT part of the string) | "cindy" "Delaware" "TOMORROW" |

Simple clauses can be connected using AND or OR.
No order of evaluation can be imposed using parenthesis.
The boolean is evaluated such that AND has higher operator precedence than OR.
E.g. A < B AND C > D OR A < E is automatically evaluated as the following ((A < B) AND (C > D)) OR (A < E).
The user is responsible for ensuring that these rules are followed.

**Routine Name:** List Index Booleans ( IB_List() )

**Routine Number:** 4.1.1.5.2.2

## 1. System Requirements

The 'list index booleans' command will enable the user to obtain a list in the logical text file of all of the constructed Index Booleans. The list contains the boolean tag and the boolean body for each boolean.

In the Menu access, the default text editor is automatically invoked to read the logical text file. In Host Language Interface access, the result can also be in the logical text string.

## 2. System Architecture

RTI_BC_traverse

## 3. System Data Structures

### 1. Input

### 2. Output

| | | |
|---|---|---|
| val_list | LIST pointer | < each entry contains tag and body of a currently defined boolean. val_list will be NULL if there is an error. > |

## 4. System Data Flow

## 5. System Control Flow

IB_List <-- RTI_BC_traverse

## 6. Design Rationale

Search the Run-time Catalog for indexes, return the tag and body of each boolean entry found.
Note the actual return value is LIST *, a NULL terminated list of entries, containing the tag and boolean condition in string format.

## 7. Test Plan

Test for case where there are no retrieved booleans.

## 8. Issues

Must return both tags and bodies. Tags alone are meaningless to the user.

**Routine Name:** Pick Index Boolean ( IB_Pick() )

**Routine Number:** 4.1.1.5.2.3

## 1. System Requirements

The 'pick index boolean' command will enable the user to make 'current' one of the index booleans. If the index boolean does not exist, then an error message appears in the logical error file. If the chosen boolean cannot be used to search the current index, an error message appears in the logical text file.

## 2. System Architecture

parse
validate

RTI_BC_retrieve
test_compatibility

CWA_I_update(IB_current)

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| tag | TAG | < identifier for retrieved boolean > |

### 2. Output

| | |
|---|---|
| IME_OK | successful picking of boolean |
| IME_BAD_TAG | bad tag |
| IME_INCOMPATIBLE | boolean not compatible with index |
| IME_NO_CURRENT | no current index |

## 4. System Data Flow

| | |
|---|---|
| parse --> | validate |
| validate --> | RTI_BC_retrieve |
| | test_compatibility |
| | CWA_I_update |

## 5. System Control Flow

| | |
|---|---|
| I_Pick <-- | parse |
| | validate |
| | RTI_BC_retrieve |
| | test_compatibility |
| | CWA_I_update |

## 6. Design Rationale

This routine will make current the specified boolean. First it checks

that there is a current index to test compatibility with. Then it searchs the run-time boolean catalog for the given tag.
If it is found, test for the compatibility of the boolean for searching the current index (e.g. attribute names are attributes of the index, values are in the domain of values for the index).

## 7. Test Plan

Test all error codes.

## 8. Issues

**Routine Name:** Modify Index Boolean ( IB_modify() )

**Routine Number:** 4.1.1.5.2.4

## 1. System Requirements

The system will enable the user to modify a index boolean. The user receives the current index boolean definition in the logical text file for modifying. The user then alters the definition and the system replaces the old definition with the new one.

In the Menu access, the default text editor is automatically invoked to read the logical text file. In Host Language Interface access, the result can also be in the logical text string.

## 2. System Architecture

| | |
|---|---|
| form_modify | ( give it the current boolean definition ) |
| parse | |
| validate | |
| test_syntax | |
| test_compatibility | |
| RTI_BC_update | |

## 3. System Data Structures

### 1. Input

### 2. Output

| | |
|---|---|
| IME_OK | success |
| IME_FAILURE | general error |
| IME_INCOMPATIBLE | boolean incompatible with current index |
| IME_BAD_BOOL | boolean syntactically invalid |
| IME_NO_CURRENT | no current index to test compatibility against |

## 4. System Data Flow

| | | |
|---|---|---|
| form_modify | --> | parse |
| parse | --> | validate |
| validate | --> | test_syntax |
| | | test_compatibility |

RTI_BC_update

## 5. System Control Flow

| | |
|---|---|
| IB_modify <-- | form_modify |
| | parse |
| | validate |
| | test_syntax |
| | test_compatibility |

## 6. Design Rationale

This routine will update the current definition for the current index boolean. First it will check that there is a current index, in order to test for compatibility. Next it checks that the user is not trying to modify the default boolean "TRUE". Then it will check that the current index boolean exists. The user is given a copy of the current boolean definition and is allowed to modify it through form_modify. The new definition is then tested for syntatic correctness and compatibility with the current index. If it is compatible, then the old definition is replaced with the new one. If it is not compatible, no change is made.
See Section 4.1.1.5.2.1 for valid syntax of boolean conditions.

## 7. Test Plan

Test all error codes.

## 8. Issues

This routine will pass to form_modify the current select definition and receive back the modified select definition in string format.

**Routine Name:** Drop Index Boolean ( IB_drop() )

**Routine Number:** 4.1.1.5.2.5

## 1. System Requirements

The system enables the user to delete a index boolean. If the index boolean does not exist, then an error message appears in the logical error file.

## 2. System Architecture

| | |
|---|---|
| parse | |
| validate | |
| RTI_BC_delete | (deallocate_WA) |
| CWA_I_update(IB_current = "TRUE") | ( if dropped is current ) |

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| tag | TAG | < tag for index boolean > |

### 2. Output

| | |
|---|---|
| IME_OK | success |
| IME_FAILURE | general error |
| IME_BAD_TAG | tag specified does not exist |

## 4. System Data Flow

| | |
|---|---|
| parse --> | validate |
| validate --> | RTI_BC_delete |
| | CWA_I_update |

## 5. System Control Flow

| | |
|---|---|
| IB_drop <-- | parse |
| | validate |
| | RTI_BC_delete |
| | CWA_I_update |

## 6. Design Rationale

This routine will drop any index boolean that is currently in the CWA. The user is not allowed to delete the default boolean, so there is a check to make sure the user is not trying to delete this boolean. It will then delete the catalog entry for the index boolean. If the index boolean that was deleted was the current index boolean then the current index boolean is updated to equal the default index boolean "TRUE".

## 7. Test Plan

Test all error codes.

## 8. Issues

**Routine Name**: Build Index Select ( IS_build() )

**Routine Number**: 4.1.1.5.3.1

## 1. System Requirements

The system will enable the user to build an index select for navigating through the index rows. The user receives in the logical text file an index select form for building such a select. The user completes the form and submits it. The system assigns the index select a tag. If an error in building the select is made, then an error is returned.

In Menu and Command access, the form is input/ouput through the logical text file. In the Menu access, the default text editor is automatically invoked to build the form. In Host Language Interface access, the form can be built through the logical text string.

## 2. System Architecture

form_new
parse
validate
assign_tag
RTI_SC_insert

## 3. System Data Structures

### 1. Input

### 2. Output

| | | |
|---|---|---|
| tag | TAG | tag of the index select created. NULL if error. |

## 4. System Data Flow

| | | |
|---|---|---|
| form_new | --> | parse |
| parse | --> | validate |
| validate | --> | assign_tag |
| | | RTI_SC_insert |

## 5. System Control Flow

| | |
|---|---|
| IS_build <-- | form_new |
| | parse |
| | validate |
| | assign_tag |
| | RTI_SC_insert |

## 6. Design Rationale

This routine will build an index select and insert it into the catalog.

The user is given a form to complete defining the index select. This definition is then parsed into an internal format. A catalog entry is created and a tag is assigned to the select definition. There is compatibility check done at this time. That is done at the time the user 'picks' the index select as current. The tag is returned. If an error is encountered then NULL is returned.

## 7. Test Plan

## 8. Issues

This routine receives it's input from the form process that is spawned. The form process will give this routine's input in string format.

| | | |
|---|---|---|
| select_list | string | < list of the attributes of the select > |

It is up to us to parse the string into our internal format.

The characters accepted as part of an attribute name are A-Z, a-z, . - * (to allow for select *). Any other character or sequence of characters can be used as a separator, as long as there are no interspersed spaces.
i.e.

| | |
|---|---|
| a : b is acceptable. | (separator = ":") |
| a :,# b is acceptable. | (separator = ":,#") |
| a : , b is not acceptable because the separator is ":" and it tries to parse "," as an attribute name. | |
| a b is acceptable. | (separator doesn't exist, but that's ok) |

**Routine Name:** List Index Selects ( IS_list() )

**Routine Number:** 4.1.1.5.3.2

## 1. System Requirements

The system will enable the user to obtain a list in the logical text file of the constructed index selects for the index.

In the Menu access the default text editor is automatically invoked to read the logical text file. In Host Language Interface access, the result can also be in the logical text string.

## 2. System Architecture

RTI_SC_traverse

## 3. System Data Structures

### 1. Input

### 2. Output

| | | |
|---|---|---|
| select_list | LIST pointer | < array of tag and definitions<br>NULL if error > |

## 4. System Data Flow

## 5. System Control Flow

IS_list <-- RTI_SC_traverse

## 6. Design Rationale

This routine will return a list of tags and definitions for all index selects currently defined in the current working area. This information is retrieved from the index select run-time catalog. If there does not currently exist any index selects, then NULL is returned. The actual type of what is returned is LIST * (a NULL-terminated list of index select information).

## 7. Test Plan

Test the listing of index selects when none are defined, and also when at least one is defined.

## 8. Issues

**Routine Name:** Pick Index Select ( IS_pick() )

**Routine Number:** 4.1.1.5.3.3

## 1. System Requirements

The system will enable the user to make as "current" one of the index selects. If the index select does not exist, an error message appears in the logical error file.

## 2. System Architecture

parse
validate
test_compatibility
CWA_I_update(IS_current)

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| tag | TAG | < tag of the index select to pick > |

### 2. Output

| | |
|---|---|
| IME_OK | success |
| IME_FAILURE | general error |
| IME_BAD_TAG | invalid tag |
| IME_INCOMPATIBLE | select to pick is incompatible with current index |
| IME_NO_CURRENT | no current index to test compatibility with |

## 4. System Data Flow

| | |
|---|---|
| parse --> | validate |
| validate --> | test_compatibility |
| | CWA_I_update |

## 5. System Control Flow

| | |
|---|---|
| IS_pick <-- | parse |
| | validate |
| | test_compatibility |
| | CWA_I_update |

## 6. Design Rationale

This routine will update the current index select. First it checks that there is a current index to verify compatibility. Next it checks that the tag given is a valid index select tag, and that the select currently exists. Next the compatibility is checked against the current index. The current working area index select variable is updated.

## 7. Test Plan

Test all error codes.

## 8. Issues

**Routine Name:** Modify Index Select ( IS_modify() )

**Routine Number:** 4.1.1.5.3.4

## 1. System Requirements

The system will enable the user to modify a index select. The user receives the current index select definition in the logical text file for modifying. He then alters the definition and the system replaces the old definition with the new one.

In the Menu access, the default text editor is automatically invoked to read the logical text file. In Host Language Interface access, the result can also be in the logical text string.

## 2. System Architecture

| | |
|---|---|
| form_modify | ( give it the current select definition ) |
| parse | |
| validate | |
| test_compatibility | |
| RTI_SC_update | |

## 3. System Data Structures

### 1. Input

### 2. Output

| | |
|---|---|
| IME_OK | success |
| IME_FAILURE | general error |
| IME_INCOMPATIBLE | select definition incompatible with current index |
| IME_BAD_SELECT | modified definition is syntactically incorrect |

## 4. System Data Flow

| | | |
|---|---|---|
| form_modify | --> | parse |
| parse | --> | validate |
| validate | --> | test_compatibility |
| | | RTI_SC_update |

## 5. System Control Flow

| | |
|---|---|
| IS_modify <-- | form_modify |
| | parse |
| | validate |
| | test_compatibility |
| | RTI_SC_update |

## 6. Design Rationale

This routine will update the current definition for the current index select.

There is a default index select "*" to select all attributes that the user is not allowed to modify. First it will check that the current index select exists, and can be modified. The user is given a copy of the current select definition and is allowed to modify it through form_modify. The new definition is then tested for validity against the current index (both syntactically and semantically). If it is compatible, then the old definition is replaced with the new one. If it is not compatible, no change is made. See Section 4.1.1.5.3.1 for definition of acceptable select input.

## 7. Test Plan

Test all error codes.

## 8. Issues

This routine will pass to form_modify the current select definition and receive back the modified select definition in string format.

**Routine Name:** Drop Index Select ( IS_drop() )

**Routine Number:** 4.1.1.5.3.5

## 1. System Requirements

The system enables the user to delete an index select. If the index select does not exist, then an error message appears in the logical error file.

## 2. System Architecture

parse
validate
RTI_SC_delete
CWA_I_update(IS_current = "*") ( if dropped is current )

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| tag | TAG | < tag for index select > |

### 2. Output

| | |
|---|---|
| IME_OK | success |
| IME_FAILURE | general error |
| IME_BAD_TAG | tag specified does not exist |

## 4. System Data Flow

| | |
|---|---|
| parse --> | validate |
| validate --> | RTI_SC_delete |
| | CWA_I_update |

## 5. System Control Flow

| | |
|---|---|
| IS_drop <-- | parse |
| | validate |
| | RTI_SC_delete |
| | CWA_I_update |

## 6. Design Rationale

This routine will drop any index select that is currently in the CWA. The user is not allowed to delete the default select, so there is a check to make sure the user is not trying to delete this select. It will then delete the catalog entry for the index select. If the index select that was deleted was the current index select then the current index select is updated to equal the default index select "*". If the tag was invalid, an error is returned.

## 7. Test Plan

Test all error codes.

## 8. Issues

**Routine Name**: Bind Column Index ( I_bind_column() )

**Routine Number**: 4.1.1.7.1

## 1. System Requirements

The system will enable the user to bind a program variable with a column of the index. He provides the name fo the column, a pointer to the program variable, the type of the variable, and the length of the program variable. When the database is being navigated, the system places the column into the program variable.

## 2. System Architecture

parse
validate
test_compatibility
fill_I_bind_struct
CWA_I_update(I_binding = TRUE)

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| attribute_name | string | < name of the attribute to bind > |
| variable_ptr | string | < pointer to some variable > |
| variable_type | string | < type of variable to bind to > |
| variable_length | int | < length of variable to bind to > |

### 2. Output

| | |
|---|---|
| IME_OK | success |
| IME_FAILURE | general error |
| IME_NO_CURRENT | no current index to validate binding |
| IME_BAD_ATTR | invalid attribute name for current_index |
| IME_BAD_TYPE | type does not match attribute in index |

## 4. System Data Flow

| | | |
|---|---|---|
| parse | --> | validate |
| validate | --> | test_compatibility |
| | | fill_I_bind_struct |
| | | CWA_I_update |

## 5. System Control Flow

| | |
|---|---|
| I_bind_column <-- | parse |
| | validate |
| | test_compatibility |
| | fill_I_bind_struct |
| | CWA_I_update |

## 6. Design Rationale

This routine will set up for binding tuple values to program variables. First it verifies that only one column is specified. Then it checks that the column name given is a valid column name for the current index, that the variable_type matches the type of the column in the index. This information is then entered into the binding structure, and the current working area is updated to reflect that there is binding for the current index. During navigation, the values of tuples are copied into the program variables.

## 7. Test Plan

Test all error codes.

## 8. Issues

The bind column command must be executed after the user has picked the corresponding index. It is at the time the user defines the bind (i.e. here) that a compatibility check is done.

**Routine Name**: Bind Table Index ( I_bind_table() )

**Routine Number**: 4.1.1.7.2

## 1. System Requirements

The system will enable the user to bind a set of program variables with a set of columns of the current index. The user makes current the index. He then provides a pointer to a data structure containing names of the columns, pointers to the program variables, type of the variables, and the lengths of the program variables. When the database is being navigated, the system places the columns into the program variables.

## 2. System Architecture

parse
validate
test_compatibility
fill_I_bind_struct
CWA_I_update(I_binding = TRUE)

## 3. System Data Structures

### 1. Input

| | |
|---|---|
| bind_info | variable_binding pointer |

### 2. Output

| | |
|---|---|
| IME_OK | success |
| IME_FAILURE | general error |
| IME_NO_CURRENT | no current index to validate binding |
| IME_BAD_ATTR | invalid column name for current_index |
| IME_BAD_TYPE | type does not match column in index |
| IME_BAD_BIND | other compatibility or syntax errors |

## 4. System Data Flow

| | |
|---|---|
| parse --> | validate |
| validate --> | test_compatibility |
| | fill_I_bind_struct |
| | CWA_I_update |

## 5. System Control Flow

| | |
|---|---|
| I_bind_table <-- | parse |
| | validate |
| | test_compatibility |
| | fill_I_bind_struct |
| | CWA_I_update |

## 6. Design Rationale

This routine will set up for binding tuple values to program variables. First it is checked that the column names given are a valid column names for the current Index, that the variable_types matche the type of the columns in the index. This information is then entered into the binding structure, and the current working area is updated to reflect that there is binding for the current index. During navigation, the values of tuples are copied into the program variables.

## 7. Test Plan

Test all error codes.

## 8. Issues

The bind table command must be executed after the user has picked the corresponding index. It is at the time the user defines the bind (i.e. here) that a compatibility check is done.

**Routine Name:** Create Indexset ( ISET_create() )

**Routine Number:** 3.1.2.1.1

## 1. System Requirements

The create indexset command allows one to create the directory, databook, catalog, and log devices of the indexset objects.

Example:
```
create indexset library
directory:
{device = [usr.smith.library]library.dat1,
 initsize = 4096,
 incrsize = 1096,
 maxsize = 200000},
databook:
{device = [usr.smith.library]library.dat1,
 initsize = 4096,
 incrsize = 1096,
 maxsize = 200000},
catalog:
{device = [usr.smith.library]library.dat1,
 initsize = 4096,
 incrsize = 1096,
 maxsize = 200000},
{device = [usr.smith.library]library.dat1,
 initsize = 4096,
 incrsize = 1096,
 maxsize = 200000},
log:
{device = [usr.smith.library]library.dat1,
 initsize = 4096,
 incrsize = 1096,
 maxsize = 200000};
```

The argument list for the previous example would look like this, where each COLUMN is one of the arguments: (Note: there can be any number of "rows" labeled 'databook', but there should be exactly two "rows" labeled 'catalog', and only one "row" labeled log. The create command must have no "rows" labeled 'directory').

| object_names | device_name | initsize | incrsize | maxsize |
|---|---|---|---|---|
| directory | [usr.smith.library]library.dat1 | 4096 | 1096 | 200000 |
| databook | [usr.smith.library]library.dat1 | 4096 | 1096 | 200000 |
| catalog | [usr.smith.library]library.dat1 | 4096 | 1096 | 200000 |
| catalog | [usr.smith.library]library.dat1 | 4096 | 1096 | 200000 |
| log | [usr.smith.library]library.dat1 | 4096 | 1096 | 200000 |

## 2. System Architecture

parse

validate

create catalogs
SOMETHING
SISET_insert

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| iset_name | string | < name of the indexset to create > |
| iset_loc | string | < location of indexset to create > |
| object_names | array of strings | < object names for each element described in corresponding locations of other arrays > |
| device_names | array of strings | < device names for each component of the indexset > |
| initsizes | array of integers | < initial sizes for each component of the indexset > |
| incrsizes | array of integers | < incremental sizes for each component of the indexset > |
| maxsizes | array of integers | < maximum sizes for each component of the indexset > |

NOTE : all of these arrays act as parallel arrays, in that
entry i of each array is related to entry i of
every other array.

### 2. Output

| | |
|---|---|
| IME_OK | ISET_create successful |
| IME_FAILURE | general error |

## 4. System Data Flow

parse --> validate
validate --> create catalogs
SOMETHING
SISET_insert
SOMETHING --> SISET_insert

## 5. System Control Flow

ISET_create <-- parse
validate
SOMETHING
RTISET_update
SISET_retrieve
SISET_delete

## 6. Design Rationale

This routine will create a new indexset. It will first check that the indexset to create does not already exists. It will then do SOMETHING, and create an indexset catalog entry for the indexset.

## 7. Test Plan

Test the creation of an existing and nonexistent indexset. Test errors that relate to SOMETHING.

## 8. Issues

**Routine Name:** Drop Indexset ( ISET_drop() )

**Routine Number:** 3.1.8.2

## 1. System Requirements

The drop indexset command allows one to drop an indexset.

## 2. System Architecture

```
parse
validate
for each index in the catalog
   I_drop        { drop the index }
end loop

remove catalog

for each set in the catalog
   S_drop_disk
end loop

remove catalog

SISET_delete
```

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| iset_name | string | < name of indexset to drop > |
| iset_loc | string | < location of indexset to drop > |

### 2. Output

| | |
|---|---|
| IME_OK | ISET_drop successful |
| IME_FAILURE | general failure |
| IME_DNE | indexset to drop does not exist |

## 4. System Data Flow

| | |
|---|---|
| parse --> | validate |
| validate --> | remove index loop |
| | remove index catalog |
| | remove set loop |
| | remove set catalog |
| | SISET_delete |

## 5. System Control Flow

| | |
|---|---|
| ISET_drop <-- | parse |
| | validate |
| | remove index loop |

remove index catalog
remove set loop
remove set catalog
SISET_delete

## 6. Design Rationale

This routine will check that the indexset to drop exists, and is not the current indexset. It will then remove each index (and potentially any associated index_kits), and each set, and then delete the index and set catalogs belonging to the indexset. Finally it will remove the system catalog entry.

Note: Cannot drop the current indexset.

## 7. Test Plan

Test all error codes.

## 8. Issues

Should ISET_drop cascade down to index_kit deletion ...NO!!

**Routine Name:** Alter Indexset ( ISET_alter() )

**Routine Number:** 3.1.3.2.5

## 1. System Requirements

The alter indexset command allows one to alter the databook, catalog, and log devices of the indexset objects. The directory devices cannot be altered without dropping the indexset.

Example:

```
alter indexset library
databook:
{device = [usr.smith.library]library.dat1,
 initsize = 4096,
 incrsize = 1096,
 maxsize = 200000},
{device = [usr.smith.library]library.dat2,
 initsize = 4096,
 incrsize = 1096,
 maxsize = 200000},
catalog:
{device = [usr.smith.library]library.icat,
 initsize = 4096,
 incrsize = 1096,
 maxsize = 200000},
{device = [usr.smith.library]library.scat,
 initsize = 4096,
 incrsize = 1096,
 maxsize = 200000},
log:
{device = [usr.smith.library]library.log,
 initsize = 4096,
 incrsize = 1096,
 maxsize = 200000};
```

My guess is that you can only alter the device and initsize if the object is empty, but you can modify the incrsize anytime, and the maxsize as long as the current size is less than the proposed maxsize.

The argument list for the previous example would look like this, where each COLUMN is one of the arguments: (Note: there can be any number of "rows" labeled 'databook', but there should be no more than two "rows" labeled 'catalog', and only one "row" labeled 'log'. The alter command must have no "rows" labeled 'directory').

| object_names | device_name | initsize | incrsize | maxsize |
|---|---|---|---|---|
| databook | [usr.smith.library]library.dat1 | 4096 | 1096 | 200000 |
| databook | [usr.smith.library]library.dat2 | 4096 | 1096 | 200000 |
| catalog | [usr.smith.library]library.icat | 4096 | 1096 | 200000 |
| catalog | [usr.smith.library]library.scat | 4096 | 1096 | 200000 |
| log | [usr.smith.library]library.log | 4096 | 1096 | 200000 |

## 2. System Architecture

parse
validate

SOMETHING
RTISET_update
SISET_retrieve /* retrieve the catalog entry */
SISET_delete /* delete the old catalog entry */
SISET_insert /* insert the updated catalog entry */

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| iset_name | string | < name of the indexset to alter > |
| iset_loc | string | < location of indexset to alter > |
| object_names | array of strings | < object names for each element described in corresponding locations of other arrays > |
| device_names | array of strings | < device names for each component of the indexset > |
| initsizes | array of integers | < initial sizes for each component of the indexset > |
| incrsizes | array of integers | < incremental sizes for each component of the indexset > |
| maxsizes | array of integers | < maximum sizes for each component of the indexset > |

NOTE: all of these arrays act as parallel arrays, in that
entry i of each array is related to entry i of
every other array.

### 2. Output

| | |
|---|---|
| IME_OK | ISET_alter successful |
| IME_FAILURE | general error |

## 4. System Data Flow

parse --> validate
validate --> SOMETHING
RTISET_update
SISET_retrieve
SISET_delete
SISET_insert
SOMETHING --> RTISET_update
SISET_retrieve
SISET_delete

SISET_insert

## 5. System Control Flow

ISET_alter <-- parse
validate
SOMETHING
RTISET_update
SISET_retrieve
SISET_delete
SISET_insert

## 6. Design Rationale

This routine will alter the indexset. It will first check that the indexset to alter already exists. It will then do SOMETHING, and update the indexset catalog entry for the indexset.

## 7. Test Plan

Test the altering of an existing and nonexistent indexset. Test errors that relate to SOMETHING.

## 8. Issues

When altering indexsets, must you respecify EVERYTHING that was originally specified when the indexset was created? If not, then how do you know, for example, which databook element they are implying to alter (for example if you change the name).

**Routine Name:** Move Indexset ( ISET_move() )

**Routine Number:** 3.1.5.2

## 1. System Requirements

The move indexset command allows one to move an indexset from one user to another.

## 2. System Architecture

```
parse
validate

check_uniqueness
for each index in the index catalog
   I_move                  { not I_move exactly, but move to new location }
for each set in the set catalog
   S_move                  { not S_move exactly, but move to new location }
SISET_retrieve             { from old user indexset catalog }
SISET_delete               { from old user indexset catalog }
SISET_insert               { to new user indexset catalog }
```

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| iset_name | string | < name of indexset to move > |
| iset_loc_old | string | < user currently owning indexset > |
| iset_loc_new | string | < user to move indexset to > |

### 2. Output

| | |
|---|---|
| IME_OK | ISET_move successful |
| IME_FAILURE | general error |
| IME_DNE | indexset to move does not exist |
| IME_NONUNIQUE | user to move indexset to already has indexset with the same name |

## 4. System Data Flow

```
parse    -->   validate
validate -->   check_uniqueness
               move index loop
               move set loop
               SISET_retrieve
               SISET_delete
               SISET_insert
```

## 5. System Control Flow

```
ISET_move <--  parse
               validate
```

check_uniqueness
move index loop
move set loop
SISET_retrieve
SISET_delete
SISET_insert

## 6. Design Rationale

This routine will move an indexset from one user to another. First it will check that the indexset to move exists. Next, it will check that the new user does not already have an indexset with the same name. Then for each index and set in the indexset, it will move them to the indexset at the new user's location. It will delete the catalog entry from the old user's system indexset catalog, and insert a catalog entry into the new user's system indexset catalog.

## 7. Test Plan

Test all error codes.

## 8. Issues

What should happen with associated index_kits when indexes are moved to different users? I think they should move too.

**Routine Name:** Copy Indexset ( ISET_copy() )

**Routine Number:** 3.1.4.2.1

## 1. System Requirements

The copy indexset command allows the user to make identical copies of indexsets.

## 2. System Architecture

[ B = copy(A) ]

parse
validate
ISET_create(B)

for each index in the index catalog
  I_copy(A.index,B.index)

for each set in the set catalog
  S_copy(A.set,B.set)

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| iset_name1 | string | < name of indexset to copy > |
| iset_loc1 | string | < location of indexset to copy > |
| iset_name2 | string | < name of indexset to copy into > |
| iset_loc2 | string | < location of indexset to copy into > |

### 2. Output

| | |
|---|---|
| IME_OK | ISET_copy successful |
| IME_FAILURE | general error |
| IME_DNE | indexset to copy does not exist |
| IME_NONUNIQUE | indexset to copy into already exists |

## 4. System Data Flow

| | | |
|---|---|---|
| parse | --> | validate |
| validate | --> | ISET_create |
| | | index copy loop |
| | | set copy loop |

## 5. System Control Flow

| | |
|---|---|
| ISET_copy <-- | parse |
| | validate |
| | ISET_create |
| | index copy loop |
| | set copy loop |

## 6. Design Rationale

This routine will make an identical copy of an indexset. First it checks that the indexset to copy exists, and that the indexset to create does not already exist. It then will copy each index from one indexset to the other. The ISET_create command will get ALL of its input from the indexset to copy from.

## 7. Test Plan

Test all error codes.

## 8. Issues

**Routine Name:** Intersect Indexset ( ISET_intersect() )

**Routine Number:** 3.1.4.2.4

## 1. System Requirements

The intersect indexset command allows one to intersect the different parts of indexsets.

## 2. System Architecture

[ C = A intersect B ]

parse
validate

ISET_create(C)

for each index in the index catalog of A
  if there is a compatible index in the index catalog of B
    I_intersect(A.index,B.index,C.index)
end loop

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| iset_name1 | string | < name of indexset to intersect > |
| iset_loc1 | string | < location of indexset to intersect > |
| iset_name2 | string | < name of indexset to intersect > |
| iset_loc2 | string | < loc of indexset to intersect > |
| iset_name3 | string | < name of indexset to intersect into > |
| iset_loc3 | string | < loc of indexset to intersect into > |

### 2. Output

| | |
|---|---|
| IME_OK | ISET_intersect successful |
| IME_FAILURE | general error |
| IME_DNE | indexset to intersect does not exist |
| IME_NONUNIQUE | indexset to create as result already exists |

## 4. System Data Flow

| | | |
|---|---|---|
| parse | --> | validate |
| validate | --> | ISET_create |
| | | intersection loop |

## 5. System Control Flow

| | | |
|---|---|---|
| ISET_intersect | <-- | parse |
| | | validate |
| | | ISET_create |
| | | intersection loop |

## 6. Design Rationale

This routine will intersect compatible indexes in the two different indexsets. First it must check that the two indexsets to intersect actually exist, and that the resulting indexset does not already exist. Next it will check for pairwise compatible indexes, each from a different indexset. If any such pairs exist, it will compute the intersection of them and place the resulting index in the newly created indexset. A catalog entry is created for the new indexset.

## 7. Test Plan

Test all error codes.

## 8. Issues

**Routine Name**: Subset Indexsets ( ISET_subset() )

**Routine Number**: 3.1.4.2.2

## 1. System Requirements

The 'subset indexset' command enables users to make indexsets from subsets of indexsets.

## 2. System Architecture

[ B = subset(A) ]
Parse
Validate

for each index in A
 I_subset(A.index, B)

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| indexset_name | string | < name of result indexset > |
| sub_indexset_name | string | < name of indexset to be subsetted > |
| select_list | string | < attribute names of indexed attrs to be subsetted > |
| bool_list | string | < string containing boolean condition to subset according to > |

### 2. Output

| | |
|---|---|
| IME_OK | successful subset indexset |
| IME_FAILURE | general failure |
| IME_NONUNIQUE | non-unique name for new/subsetted indexset or indexspace |
| IME_BAD_BOOL | boolean syntax error |

## 4. System Data Flow

parse --> validate
validate --> I_subset

## 5. System Control Flow

ISET_subset<-- parse
validate
I_subset

## 6. Design Rationale

The indexset and indexspace(s) for the result to go into must have already been created and must have the same indexspace names as the

iset to be subsetted.

## 7. Test Plan

Test Cases should check for correct handling of invalid names, bad booleans , bad selects.

## 8. Issues

**Routine Name:** Subtract Indexset ( ISET_subtract() )

**Routine Number:** 3.1.4.2.5

## 1. System Requirements

The subtract indexset command allows one to subtract the different parts of indexsets. [ Namely the indexes and sets? ]

## 2. System Architecture

[ C = A - B ]

parse
validate

ISET_create(C)

for each index in the index catalog of A
  if there is a compatible index in the index catalog of B
    I_subtract(A.index,B.index,C.index)
end loop

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| iset_name1 | string | < name of indexset to subtract > |
| iset_loc1 | string | < location of indexset to subtract > |
| iset_name2 | string | < name of indexset to subtract > |
| iset_loc2 | string | < loc of indexset to subtract > |
| iset_name3 | string | < name of indexset to subtract into > |
| iset_loc3 | string | < loc of indexset to subtract into > |

### 2. Output

| | |
|---|---|
| IME_OK | ISET_subtract successful |
| IME_FAILURE | general error |
| IME_DNE | indexset to subtract does not exist |
| IME_NONUNIQUE | indexset to create as result already exists |

## 4. System Data Flow

| | | |
|---|---|---|
| parse | --> | validate |
| validate | --> | ISET_create |
| | | subtraction loop |

## 5. System Control Flow

| | |
|---|---|
| ISET_subtract <-- | parse |
| | validate |
| | ISET_create |
| | subtraction loop |

## 6. Design Rationale

This routine will subtract compatible indexes in the two different indexsets. First it must check that the two indexsets to subtract actually exist, and that the resulting indexset does not already exist. Next it will check for pairwise compatible indexes, each from a different indexset. If any such pairs exist, it will compute the subtraction of them and place the resulting index in the newly created indexset. A catalog entry is created for the new indexset.

## 7. Test Plan

Test all error codes.

## 8. Issues

**Routine Name**: Union Indexsets ( ISET_union() )

**Routine Number**: 3.1.4.2.3

## 1. System Requirements

The 'union indexset' command enables users to union the different parts of indexsets.

## 2. System Architecture

[ C = A union B ]

parse
validate

for each pair of compatible indexes in A and B
  I_union(A.index,B.index,C.index)

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| result_indexset | string | < name of indexset to be created > |
| un_indexset_name1 | string | < name of indexset to be unioned > |
| un_indexset_name2 | string | < name of indexset to be unioned > |

### 2. Output

| | |
|---|---|
| IME_OK | ISET_union successful |
| IME_FAILURE | general failure |
| IME_NONUNIQUE | non-unique name for new indexset or indexspace |
| IME_DNE | indexset to union does not exist |

## 4. System Data Flow

| | | |
|---|---|---|
| parse | --> | validate |
| validate | --> | I_union |

## 5. System Control Flow

| | |
|---|---|
| ISET_union <-- | parse |
| | validate |
| | I_union |

## 6. Design Rationale

The result indexset must have already been created, with indexspaces matching those in indexset 1 being unioned.

For each combination of indexes from indexsets 1 and 2, union the indexes.

## 7. Test Plan

Test Cases should check for correct handling of invalid names.

## 8. Issues

If none of the indexes in either indexset are compatible for union, what should the resulting indexset look like?
--> No indexes in it, but indexspaces matching indexset 1 being unioned.

How do we know what indexspace to put the index in?
--> Put it in the one corresponding to the one in indexset 1 (indexes will have same names, indexspaces but different isets)

What happens when we run out of space in the ispace?

**Routine Name:** Create Indexspace ( ISPACE_create() )

**Routine Number:** 3.1.2.1.2

## 1. System Requirements

The create indexspace command allows one to create the databook devices associated with the indexspace.
Example:
create indexspace library
databook:
{device = [usr.smith.library]library.dat1,
initsize = 4096,
incrsize = 1096,
maxsize = 200000},
{device = [usr.smith.library]library.dat2,
initsize = 4096,
incrsize = 1096,
maxsize = 100000};

The argument list for the previous example would look like this, where each COLUMN is one of the arguments:

| device_name | initsize | incrsize | maxsize |
|---|---|---|---|
| [usr.smith.library]library.dat1 | 4096 | 1096 | 200000 |
| [usr.smith.library]library.dat2 | 4096 | 1096 | 100000 |

## 2. System Architecture

parse
validate

SOMETHING
SISET_insert

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| ispace_name | string | < name of indexspace to create > |
| iset_name | string | < name of indexset to create indexspace in > |
| iset_loc | string | < location of indexset > |
| device_names | array of strings | < device names for indexset components > |
| initsizes | array of integers | < initial sizes for indexset components > |
| incrsizes | array of integers | < increment sizes for indexset components > |
| maxsizes | array of integers | < maximum sizes for indexset components > |

### 2. Output

| | |
|---|---|
| IME_OK | ISPACE_create successful |
| IME_FAILURE | general error |

## 4. System Data Flow

```
parse          --> validate
validate       --> SOMETHING
                   SISET_insert
SOMETHING      --> SISET_insert
```

## 5. System Control Flow

```
ISPACE_create <--  parse
                   validate
                   SOMETHING
                   SISET_insert
```

## 6. Design Rationale

This routine will create the indexspace within an indexset. It will first check that there does not already exist an indexspace with the same name in the indexset. It will then do SOMETHING, and insert an entry into the indexset catalog expressing the creation of an indexspace.

## 7. Test Plan

Test all error codes.

## 8. Issues

What is this routine supposed to do?

**Routine Name:** Alter Indexspace ( ISPACE_alter() )

**Routine Number:** 3.1.3.2.4

## 1. System Requirements

The alter indexspace command allows one to alter the databook devices associated with the indexspace.
Example:

```
alter indexspace library
databook:
{device = [usr.smith.library]library.dat1,
 initsize = 4096,
 incrsize = 1096,
 maxsize = 200000},
{device = [usr.smith.library]library.dat2,
 initsize = 4096,
 incrsize = 1096,
 maxsize = 100000};
```

The argument list for the previous example would look like this, where each COLUMN is one of the arguments:

| device_name | initsize | incrsize | maxsize |
|---|---|---|---|
| [usr.smith.library]library.dat1 | 4096 | 1096 | 200000 |
| [usr.smith.library]library.dat2 | 4096 | 1096 | 100000 |

## 2. System Architecture

parse
validate

SOMETHING
SISET_update

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| iset_name | string | < name of the indexset to alter indexspace > |
| iset_loc | string | < location of indexset > |
| device_names | array of strings | < device names for indexset components > |
| initsizes | array of integers | < initial sizes for indexset components > |
| incrsizes | array of integers | < increment sizes for indexset components > |
| maxsizes | array of integers | < maximum sizes for indexset components > |

### 2. Output

| | |
|---|---|
| IME_OK | ISPACE_alter successful |
| IME_FAILURE | general error |

## 4. System Data Flow

| | |
|---|---|
| parse | --> validate |
| validate | --> SOMETHING |
| | SISET_update |
| SOMETHING | --> SISET_update |

## 5. System Control Flow

| | |
|---|---|
| ISPACE_alter <-- | parse |
| | validate |
| | SOMETHING |
| | SISET_update |

## 6. Design Rationale

This routine will alter the indexspaces within an indexset. It will first check that the indexspaces within that indexset already exist in order to alter them. It will then do SOMETHING, and update the indexset catalog entry, updating the indexspace information within the indexset catalog entry.

## 7. Test Plan

Test all error codes.

## 8. Issues

What is this routine supposed to do?

**Routine Name:** Create Index Kit ( IK_create() )

**Routine Number:** 3.1.2.5

## 1. System Requirements

The 'create indexkit' command allows one to create indexkit objects.

## 2. System Architecture

parse
validate

SIK_insert

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| indexkit | string | < name of indexkit to create > |
| kitset | string | < name of kit_set to contain indexkit > |

### 2. Output

| | |
|---|---|
| IME_OK | indexkit creation successful |
| IME_FAILURE | general error |
| IME_NONUNIQUE | indexkit with same name already exists in kitset |

## 4. System Data Flow

parse --> validate
validate --> SIK_insert

## 5. System Control Flow

IK_create <-- parse
validate
SIK_insert

## 6. Design Rationale

This routine will create an empty catalog entry for an indexkit.
This routine will check that an indexkit with the same name does not already exist in the kitset. The only field that will have any value is the indexkit name. All other catalog fields are changed using the IK_update command. (See design 3.1.3.5).

## 7. Test Plan

Test all error codes.

## 8. Issues

**Routine Name:** Drop Index Kit ( IK_drop() )

**Routine Number:** 3.1.8.5

## 1. System Requirements

The 'drop indexkit' command allows one to drop an indexkit.

## 2. System Architecture

parse
validate

component_update < update the component catalogs if necessary >

SIK_delete

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| indexkit | string | < indexkit to drop > |
| kitset | string | < kitset containing indexkit > |

### 2. Output

| | |
|---|---|
| IME_OK | successful dropping of indexkit |
| IME_FAILURE | general error |
| IME_DNE | indexkit to drop does not exist |

## 4. System Data Flow

| | |
|---|---|
| parse --> | validate |
| validate --> | component_update |
| | SIK_delete |

## 5. System Control Flow

| | |
|---|---|
| IK_drop <-- | parse |
| | validate |
| | Component_update |
| | SIK_delete |

## 6. Design Rationale

This routine will check that the indexkit to drop exists, and if so, will delete the system catalog entry for the indexkit. Because an indexkit is just a logical grouping of several physical objects, the actual objects are not deleted, just the logical grouping. If any of the components track indexkit associations, then those catalogs will be updated too.

## 7. Test Plan

Test all error codes.

## 8. Issues

Component catalog changes are currently only rnade for indexes.
If the other components' catalog tracks indexkit
associations, that code needs to be added.

**Routine Name**: Move Index Kit ( IK_move() )

**Routine Number**: 3.1.5.5

## 1. System Requirements

The 'move indexkit' command allows one to move an indexkit from one kitset to another.

## 2. System Architecture

parse
validate

check_uniqueness

SIK_retrieve < update both kitset catalogs >
SIK_delete
SIK_insert

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| indexkit | string | < name of indexkit to move > |
| kitset_old | string | < name of kitset to move from > |
| kitset_new | string | < name of kitset to move to > |

### 2. Output

| | |
|---|---|
| IME_OK | move indexkit successful |
| IME_FAILURE | general error |
| IME_DNE | indexkit does not exist |
| IME_NONUNIQUE | kitset to move to already has indexkit w/same name. |

## 4. System Data Flow

| | | |
|---|---|---|
| parse | --> | validate |
| validate | --> | check_uniqueness |
| | | SIK_retrieve |
| | | SIK_delete |
| | | SIK_insert |

## 5. System Control Flow

| | | |
|---|---|---|
| IK_move | --> | parse |
| | | validate |
| | | check_uniqueness |
| | | SIK_retrieve |
| | | SIK_delete |
| | | SIK_insert |

## 6. Design Rationale

This routine will move an indexkit from one kitset to another. This routine will check that the indexkit to move does in fact exist. This routine will check that the new kitset to move the indexkit to does not already have an indexkit with the same name. If it does not, it will move the indexkit files to the new kitset, and update the catalogs so that the indexkit catalog entry is in the new kitset catalog. It will also need to update the components catalog entries, if they track indexkit associations.

## 7. Test Plan

Test all error codes.

## 8. Issues

**Routine Name**: Update Index Kit (IK_update() )

**Routine Number**: 3.1.3.5

## 1. System Requirements

The 'update indexkit' command enables users to update any of the components of the indexkit individually.

## 2. System Architecture

parse
validate

update component catalog
update system catalog for indexkits

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| kitset | string | < name of kitset for kit > |
| indexkit | string | < name of the kit to be altered > |
| component | string | < name of the component to update > |
| comp_name | string | < new component value > |
| comp_set | string | < name of the component-set > |

### 2. Output

| | |
|---|---|
| IME_OK | < successful update > |
| IME_FAILURE | < general failure > |
| IME_DNE | < indexkit, kitset, or component non-existent > |

## 4. System Data Flow

| | |
|---|---|
| parse --> | validate |
| validate --> | update_comp_cat |
| | SIKC_update |

## 5. System Control Flow

IK_update <-- parse
validate
update_comp_cat
SIKC_update

## 6. Design Rationale

This routine provides a means for associating a component with an indexkit. The name and set for the component is given and the Indexkit is updated by making calls to update the right catalogs. Issuing this command more than once will result in changing the component from what it was before to the new value.

## 7. Test Plan

Test for invalid component, set, and kit names.

## 8. Issues

Unique identifier for an indexkit is the kitname and the kitset-name.

Currently, this routine will update the system catalog for indexkits, for all component changes, but it will only change the component's catalog for indexes. If the other components' catalog tracks indexkit associations, that code needs to be added.

There is currently no check to make sure that the component name that a component is being updated to actually exists.
The index component checks for existence by default, because it must exist in order to update the index component's catalog entry.

**Routine Name:** Copy Index Kit ( IK_copy() )

**Routine Number:** 3.1.4.5.1

## 1. System Requirements

The 'copy indexkit' command allows one to make identical copies of indexkits.

## 2. System Architecture

[ B = copy(A) ]

parse
validate

IK_create(B)
MAN_copy(A.intro,B.intro)
IK_update(B.intro)
I_copy(A.index,B.index)
IK_update(B.index)
DBC_copy(A.dict,B.dict)
IK_update(B.dict)
BK_copy(A.thes,B.thes)
IK_update(B.thes)

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| indexkit1 | string | < name of indexkit to copy > |
| kitset1 | string | < kitset containing 'indexkit1' > |
| indexkit2 | string | < name of indexkit to create copy into > |
| kitset2 | string | < kitset to contain new indexkit > |

### 2. Output

| | |
|---|---|
| IME_OK | successful copy of indexkit |
| IME_FAILURE | general error |
| IME_DNE | 'indexkit1' does not exist |
| IME_NONUNIQUE | 'indexkit2' already exists |

## 4. System Data Flow

| | |
|---|---|
| parse --> | validate |
| validate --> | IK_create |
| | MAN_copy |
| | IK_update |
| | I_copy |
| | IK_update |
| | DBC_copy |
| | IK_update |
| | BK_copy |

IK_update

## 5. System Control Flow

IK_copy <-- parse
validate
IK_create
MAN_copy
IK_update
I_copy
IK_update
DBC_copy
IK_update
BK_copy
IK_update

## 6. Design Rationale

This routine will create an identical copy of an indexkit. First it will check that the indexkit to copy does in fact exist, and then it will check that the indexkit to create does not already exist. It will try to create the new indexkit. Then the introduction, index, dictionary, and thesaurus components of the indexkit are copied, and a catalog entry in the indexkit is created.

## 7. Test Plan

Test all error codes.

## 8. Issues

The objects that result from copying the different components of the indexkit are automatically placed in the 'set' associated with the first kit's components.

**Routine Name:** Intersect Index Kit ( IK_intersect() )

**Routine Number:** 3.1.4.5.4

## 1. System Requirements

The 'intersect indexkit' command allows one to intersect the different parts of indexkits.

## 2. System Architecture

[ C = A intersect B ]

parse
validate

IK_create(C)

| | |
|---|---|
| MAN_intersect(A.intro, B.intro, C.intro) | < manuscript intersection > |
| IK_update(C.intro) | |
| I_intersect(A.index, B.index, C.index) | < index intersection > |
| IK_update(C.index) | |
| DBC_intersect(A.dict, B.dict, C.dict) | < database intersection > |
| IK_update(C.dict) | |
| BK_intersect(A.thes, B.thes, C.thes) | < book intersection > |
| IK_update(C.thes) | |

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| indexkit1 | string | < name of indexkit to intersect > |
| kitset1 | string | < kitset containing 'indexkit1' > |
| indexkit2 | string | < name of indexkit to intersect > |
| kitset2 | string | < kitset containing 'indexkit2' > |
| indexkit3 | string | < name of indexkit to contain result > |
| kitset3 | string | < kitset to contain 'indexkit3' > |

### 2. Output

| | |
|---|---|
| IME_OK | successful intersection |
| IME_FAILURE | general error |
| IME_DNE | 'indexkit1' or 'indexkit2' does not exist |
| IME_NONUNIQUE | 'indexkit3' already exists |

## 4. System Data Flow

| | |
|---|---|
| parse --> | validate |
| validate --> | IK_create |
| | MAN_intersect |
| | IK_update |
| | I_intersect |
| | IK_update |

DBC_intersect
IK_update
BK_intersect
IK_update

## 5. System Control Flow

IK_intersect <-- parse
validate
IK_create
MAN_intersect
IK_update
I_intersect
IK_update
DBC_intersect
IK_update
BK_intersect
IK_update

## 6. Design Rationale

This routine will make sure that the indexkit to create as the result does not already exist, and that the two indexkits to intersect do already exist. It will then create the new indexkit, and then proceed to intersect corresponding parts of each indexkit.

## 7. Test Plan

Test all error codes.

## 8. Issues

The objects that result from intersecting the different components of the indexkits are automatically placed in the 'set' associated with the first kit's components.

**Routine Name**: Subset Index Kit ( IK_subset() )

**Routine Number**: 3.1.4.5.2

## 1. System Requirements

The 'subset indexkit' command enables users to make indexkits from subsets of indexkits.

## 2. System Architecture

[ B = subset(A) ]

parse
validate

IK_create(B)

MAN_subset(A.intro,B.intro)
IK_update(B.intro)

I_subset(A.index,B.index)
IK_update(B.index)

DBC_subset(A.dict,B.dict)
IK_update(B.dict)

BK_subset(A.thes,B.thes)
IK_update(B.thes)

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| indexkit1 | string | < name of indexkit to be subsetted > |
| kitset1 | string | < kitset containing 'indexkit1' > |
| indexkit2 | string | < name of resulting indexkit > |
| kitset2 | string | < kitset to contain 'indexkit2' > |
| bool_cond | string | < boolean condition for subset > |

### 2. Output

| | |
|---|---|
| IME_OK | successful creation of subset indexkit |
| IME_FAILURE | general failure |
| IME_NONUNIQUE | 'indexkit2' already exists |
| IME_DNE | 'indexkit1' does not exist |

## 4. System Data Flow

| | | |
|---|---|---|
| parse | --> | validate |
| validate | --> | IK_create |
| | | MAN_subset |
| | | IK_update |

I_subset
IK_update
DBC_subset
IK_update
BK_subset
IK_update

## 5. System Control Flow

IK_subset <-- parse
validate
IK_create
MAN_subset
IK_update
I_subset
IK_update
DBC_subset
IK_update
BK_subset
IK_update

## 6. Design Rationale

A new indexkit is created. Objects for the new kit are created by subsetting the objects of the original indexkit according to the boolean condition.
The new indexkit is updated so that these newly created kit objects are associated with it.

## 7. Test Plan

Test all error codes.

## 8. Issues

The objects created by subsetting the different components of the indexkits belong to the 'sets' associated with the component being subsetted.

**Routine Name:** Subtract Index Kit ( IK_subtract() )

**Routine Number:** 3.1.4.5.5

## 1. System Requirements

The 'subtract indexkit' command allows one to subtract the different parts of indexkits.

## 2. System Architecture

[ C = A - B ]

parse
validate

IK_create(C)

| | |
|---|---|
| MAN_subtract(A.man, B.man, C.man) | < manuscript subtraction : intro > |
| I_subtract(A.index, B.index, C.index) | < index subtraction : index > |
| DBC_subtract(A.dict, B.dict, C.dict) | < db cluster subtraction : dict > |
| BK_subtract(A.thes, B.thes, C.thes) | < book subtraction : thesaurus > |

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| indexkit | string | < name of indexkit to subtract > |
| kitset1 | string | < kitset containing 'indexkit1' > |
| indexkit | string | < name of indexkit to subtract > |
| kitset2 | string | < kitset containing 'indexkit2' > |
| indexkit | string | < name of indexkit to be result > |
| kitset3 | string | < kitset to contain 'indexkit3' > |

### 2. Output

| | |
|---|---|
| IME_OK | successful indexkit subtraction |
| IME_FAILURE | general error |
| IME_DNE | 'indexkit1' or 'indexkit2' does not exist |
| IME_NONUNIQUE | 'indexkit3' already exists |

## 4. System Data Flow

| | |
|---|---|
| parse --> | validate |
| validate --> | IK_create |
| | MAN_subtract |
| | IK_update |
| | I_subtract |
| | IK_update |
| | DBC_subtract |
| | IK_update |
| | BK_subtract |
| | IK_update |

## 5. System Control Flow

IK_subtract <--parse
validate
IK_create
MAN_subtract
IK_update
I_subtract
IK_update
DBC_subtract
IK_update
BK_subtract
IK_update

## 6. Design Rationale

This routine will subtract all components of an indexkit. This routine will check that the indexkit to create does not already exist. This routine will check that the two indexkits to subtract do actually exist, and if so, will subtract corresponding parts of each indexkit, placing the result in the newly created indexkit.

## 7. Test Plan

Test all error codes.

## 8. Issues

**Routine Name**: Union Index Kit (IK_union() )

**Routine Number**: 3.1.4.5.3

## 1. System Requirements

The 'union indexkit' command enables users to union the different parts of indexkits.

## 2. System Architecture

[ C = A intersect B ]

parse
validate

IK_create(C)

MAN_union(A.intro,B.intro,C.intro)
IK_update(C.intro)

I_union(A.index,B.index,C.index)
IK_update(C.index)

DBC_union(A.dict,B.dict,C.dict)
IK_update(C.dict)

BK_union(A.thes,B.thes,C.thes)
IK_update(C.thes)

## 3. System Data Structures

### 1. Input

| | | |
|---|---|---|
| indexkit1 | string | < name of indexkit to be unioned > |
| kitset1 | string | < kitset containing 'indexkit1' > |
| indexkit2 | string | < name of indexkit to be unioned > |
| kitset2 | string | < kitset containing 'indexkit2' > |
| indexkit3 | string | < name of resulting indexkit > |
| kitset3 | string | < kitset to contain 'indexkit3' > |

### 2. Output

| | |
|---|---|
| IME_OK | successful creation of union indexkit |
| IME_FAILURE | general failure |
| IME_NONUNIQUE | 'indexkit3' already exists |
| IME_DNE | 'indexkit1' or 'indexkit2' does not exist |

## 4. System Data Flow

| | |
|---|---|
| parse --> | validate |
| validate --> | IK_create |
| | MAN_union |

IK_update
I_union
IK_update
DBC_union
IK_update
BK_union
IK_update

5. **System Control Flow**

IK_union <-- parse
validate
IK_create
MAN_union
IK_update
I_union
IK_update
DBC_union
IK_update
BK_update
IK_update

6. **Design Rationale**

Create a new indexkit. Create each of the elements in the index kit by unioning the two objects in the original kits. Use update to associate the new elements with the new indexkit.

7. **Test Plan**

Test cases should check for correct handling of invalid names.

8. **Issues**

The objects that result from unioning the different components of the indexkits are automatically placed in the 'set' associated with the first kit's components.

# Part III: User's Guide

Although all commands and operations described in Part I of this report were designed, only a subset was implemented. Table E lists the operations that were implemented both at the Host Language Interface and at the Command Language Interface. In the rest of this section we give the User's Guide for the Command Language Interface.

## Table E: UIMS Commands

| Index Management Commands | | | |
|---|---|---|---|
| create index | drop index | insert index | copy index |
| update index | move index | delete index | search index |

| Index Browsing Commands | | |
|---|---|---|
| retrieve index | pick index | save index |
| return index | list indexes | |

| Index Navigation Commands | | |
|---|---|---|
| first in index | next in index | fetch using index |
| last in index | previous in index | |
| build index boolean | list index booleans | pick index boolean |
| modify index boolean | drop index boolean | |

| Index Run-Time Environment Commands | |
|---|---|
| create column binding | create row binding |
| drop column binding | drop row binding |

| Indexset Management Commands | |
|---|---|
| create indexset | drop indexset |

## Indexes

In this section, we discuss the use of the indexes commands.

### 1. Creating Index Objects

We specify how to create indexsets and indexes.

### 1.1. Creating Indexsets

The **create indexset** command allows one to create the directory databook, catalogue and log devices of the indexset objects. The format of the command is shown in Figure 1a. Figure 1b contains requests to create indexsets nssdca.smith.library and ipac.smith.library.

```
create indexset <indexset_name>
```

Figure 1a: Creating indexsets

```
create indexset nssdca.smith.library
create indexset ipac.smith.library
```

Figure 1b: Creating indexsets

### 1.2. Creating Indexes

The **create index** command allows one to create index objects within indexsets. The format of the command is shown in Figure 2a. "type" can be one of "*btree, heap, rtree*", while schema is a file containing the information about the attributes of the index, and is a file with the following structure

```
attribute1_name    type    length
attribute2_name    type    length
attribute3_name    type    length
      .....          ...     .....
```

"type" can be one of *string, int* and *float*. Figure 2b contains requests to create indexes subject, grid and grid-time within indexset nssdca.smith.smith.

```
create index <index_name> <indexset> <type> <schema>
```

Figure 2a: Creating indexes

```
create subject nssdca.smith.library btree btree.schema
create grid nssdca.smith.library rtree 2rtree.schema
create grid-time nssdca.smith.library rtree 1rtree.schema
```

Figure 2b: Creating indexes

where the schema files are respectively

btree.schema:

| | | |
|---|---|---|
| subject | string | 20 |
| pointer | int | 4 |

2rtree.schema:

| | | |
|---|---|---|
| lon1 | real | 8 |
| lat1 | real | 8 |
| lon1 | real | 8 |
| lat2 | real | 8 |
| pointer | int | 4 |

1rtree.schema:

| | | |
|---|---|---|
| time1 | real | 8 |
| time2 | real | 8 |
| pointer | int | 4 |

## 2. Deleting Index Objects

We specify how to delete indexsets and indexes.

### 2.1. Deleting Indexsets

The **delete indexset** command allows one to delete the indexset objects. The format of the command is shown in Figure 3a. Figure 3b contains requests to delete indexsets nssdca.smith.library and ipac.smith.library.

```
delete indexset <indexset_name>
```

Figure 3a: Deleting indexsets

```
delete indexset nssdca.smith.library
delete indexset ipac.smith.library
```

Figure 3b: Deleting indexsets

**2.2. Deleting Indexes**

The **dlete index** command allows one to delete index objects from indexsets. The format of the command is shown in Figure 4a. Figure 4b contains requests to create indexes subject, and grid within indexset nssdca.smith.smith.

```
drop index <index_name> <indexset>
```

Figure 4a: Deleting indexes

```
drop subject nssdca.smith.library
drop grid nssdca.smith.library
```

Figure 4b: Deleting indexes

**3. Modifying Index Objects**

The **update**, **insert** and **delete index** commands allow one to modify indexes. The format of the commands is shown in Figure 3a. "update" updates the current row with the values in "row". "insert" inserts a new row with values in "row" after the current row, while "delete" deletes the current row. Figure 3b contains requests to modify index subject within indexset nssdca.smith.smith.

```
update index  <row>
insert into index <row>
delete from index
```

Figure 5a: Modifying indexes

```
first open and make the index current
update index  subject|New Title|pointer|101
insert into index subject|Last Title|pointer|104
delete from index
```

Figure 5b: Modifying indexes

## 4. Reproducing Index Objects

We specify how to reproduce indexes. The reproducing commands **copy** and **move** allow one to reproduce information about one index into another index.

### 4.1. Copying Indexes

The **copy** command allows one to make identical copies of indexes. The format of the command is shown in Figure 6a. Figure 6b contains requests to copy the subject index from indexset nssdca.smith.library onto another index "newsubject" within the indexset ipac.smith.library.

```
copy index <index_name1> <indexset1>  <index_name2> <indexset2>
```

Figure 6a: Copying Indexes

```
copy index subject nssdca.smith.library newsubject ipac.smith.library
```

Figure 6b: Copying Indexes

### 4.2. Moving Indexes

The **move** command allows one to one index from one indexset to another, or within an indexset to a different name. The format of the command is shown in Figure 7a. Figure 7b contains requests to move the subject index from indexset nssdca.smith.library into another index "newsubject" within the indexset ipac.smith.library.

```
move index <index_name1> <indexset1>  <index_name2> <indexset2>
```

Figure 7a: Copying Indexes

```
move index subject nssdca.smith.library newsubject ipac.smith.library
```

Figure 7b: Copying Indexes

## 5. Searching Index Objects

We specify how to navigate through and search indexes. The navigation commands **first**, **next**, **previous**, **last**, allow one to position the cursor to the elements of an index. The command **fetch** allows one to bring in a record. Indexes are **retrieved**, in the beginning, then **saved** (if they have been modified), and finally **returned**. The commands **pick** and **list** allow one to make an index current and check which indexes are opened respectively.

### 5.1. Opening Indexes

The **retrieve** command allows one to open an index. The format of the command is shown in Figure 8a. "mode" can be one of "read-only" and "modify". "tag" is a unique identifier that the user supplies; if not present the system generates one. Figure 8b contains requests to retrieve the subject and the grid index from indexset nssdca.smith.library.

```
retrieve index <index_name1> <indexset1>  <mode> [<tag>]
```

Figure 8a: Retrieving Indexes

```
retrieve index subject nssdca.smith.library read-only
retrieve index grid nssdca.smith.library modify t2
```

Figure 8b: Retrieving Indexes

### 5.2. Listing Retrieved Indexes

The **list** command allows one to list the opened indexes and see their tags. The format of the command is shown in Figure 9a. It has no arguments. Figure 9b shows the output of the command.

```
list index
```

Figure 9a: Listing Indexes

```
INDEX:    nssdca.smith.library/subject    t1
INDEX:    nssdca.smith.library/grid       t2
```

Figure 9b: Listing Indexes

### 5.3. Picking Indexes

The **pick** command allows one to pick an opened index and make it current. The format of the command is shown in Figure 10a. "tag" is a unique identifier that the user supplies. Figure 10b contains a request to pick the subject index from indexset nssdca.smith.library.

```
pick index <tag>
```

Figure 10a: Picking Indexes


Figure 10b: Picking Indexes

**5.4. Saving Indexes**

The **save** command allows one to save permanently an opened and modified index. The format of the command is shown in Figure 11a. "tag" is a unique identifier that the user supplies. Figure 11b shows how to save the subject index in indexset nssdca.smith.library.


Figure 11a: Picking Indexes

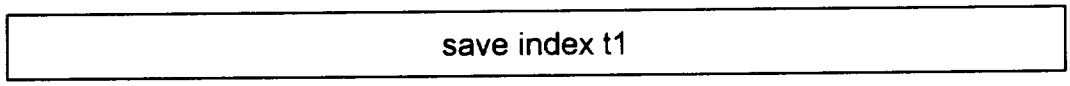


Figure 11b: Picking Indexes

**5.4. Returning Indexes**

The **return** command allows one to close an opened index. The format of the command is shown in Figure 12a. "tag" is a unique identifier that the user supplies. Figure 12b shows how to close the subject index from the indexset nssdca.smith.library.

Figure 12a: Returning Indexes

```
return index t1
```

Figure 12b: Returning Indexes

### 5.5. Positioning within Indexes

The navigation commands **first**, **next**, **previous**, **last**, allow one to position the cursor to the elements of an index. These can be executed after an index has been picked to be the current one and a boolean for the search has been defined (if no such boolean has been defined it is assumed that one want to navigate through all of the elements). The command **fetch** allows one to bring in a record.
The format of the commands is shown in Figure 13a. "tag" is a unique identifier that the user supplies. Figure 13b shows how to navigate and get records from the subject index from the indexset nssdca.smith.library.

```
first in index
next in index
previous in index
last in index
fetch from index
```

Figure 13a: Navigating Through Indexes

```
open subject index and
pick it to be the current one; then

first in index
fetch from index
next in index
fetch from index
fetch from index
last in index
fetch from index
previous in index
```

Figure 13b: Navigating Through Indexes

## 6. Manipulating Booleans

We specify how to define booleans and use them for searching through indexes.

### 6.1. Building Booleans

The **build boolean** command allows one to define booleans and use them for searching through indexes. The format of the command is shown in Figure 14a. "expression" is a boolean expression and "tag" is a unique identifier that the user supplies; if not present the system generates one. Figure 14b contains requests to build two booleans.

```
build boolean <expression> [<tag>]
```

Figure 14a: Building Booleans

```
build boolean subject="New Title"
build boolean lat1=15 b2
```

Figure 14b: Building Booleans

### 6.2. Listing Booleans

The **list boolean** command allows one to list the defined indexes and see their tags. The format of the command is shown in Figure 15a. It has no arguments. Figure 15b shows the output of the command.

```
list boolean
```

Figure 15a: Listing Booleans

```
BOOLEAN:    subject="New Title"    b1
BOOLEAN:    lat1=15                b2
```

Figure 15b: Listing Booleans

### 6.3. Picking Booleans

The **pick boolean** command allows one to pick a previously defined boolean and make it current. The format of the command is shown in Figure 16a. "tag" is a unique identifier that the user supplies. Figure 16b contains a request to pick the subject boolean.

```
pick boolean <tag>
```

Figure 16a: Picking Booleans

```
pick boolean b1
```

Figure 16b: Picking Booleans

### 6.4. Modifying Booleans

The **modify boolean** command allows one to change the expression of a defined boolean. The format of the command is shown in Figure 17a. Figure 17b shows how to modify a previously defined boolean.

```
modify boolean <new_expression>
```

Figure 18a: Modifying Booleans

```
modify boolean subject="Second Title"
modify boolean lat1=20
```

Figure 18b: Modifying Booleans

### 6.5. Droping Booleans

The **drop boolean** command allows one to drop a previously defined boolean. The format of the command is shown in Figure 19a. "tag" is a unique identifier that the user supplies. Figure 19b shows how to drop the boolean defined on "subject".


Figure 19a: Droping Booleans


Figure 19b: Droping Booleans

# Part IV: Reference Manual

Although all commands and operations described in Part I of this report were designed, only a subset was implemented. Table E (repeated from Part III) lists the operations that were implemented both at the Host Language Interface and at the Command Language Interface. In the rest of this section we give the Reference Manual for both Interfaces.

## Table E: UIMS Commands

| Index Management Commands | | | |
|---|---|---|---|
| create index | drop index | insert index | copy index |
| update index | move index | delete index | search index |

| Index Browsing Commands | | |
|---|---|---|
| retrieve index | pick index | save index |
| return index | list indexes | |

| Index Navigation Commands | | |
|---|---|---|
| first in index | next in index | fetch using index |
| last in index | previous in index | |
| build index boolean | list index booleans | pick index boolean |
| modify index boolean | drop index boolean | |

| Index Run-Time Environment Commands | |
|---|---|
| create column binding | create row binding |
| drop column binding | drop row binding |

| Indexset Management Commands | |
|---|---|
| create indexset | drop indexset |

# Host Language Interface

**Function Name:**

ISET_create

**Purpose:**

To create an indexset.

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: char *indexset The name of the indexset

**Description:**

To create an indexset

```
ISET_create(indexset)
```

**Errors:**

If there is an existing indexset with the same name

**Examples:**

```
ISET_create("iueobs");
```

**See Also:**

**Function Name:**

ISET_delete

**Purpose:**

To delete an indexset.

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: char *indexset The name of the indexset

**Description:**

To delete an indexset

```
ISET_delete(indexset)
```

**Errors:**

If there is no indexset with the given name

**Examples:**

```
ISET_delete("iueobs");
```

**See Also:**

**Function Name:**

I_create

**Purpose:**

To create an index in an indexset.

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: WA *wa A pointer to a work area
IN: char *indexname The name of the source index
IN: char *indexset The name of source parent indexset
IN: FORMAT *mode The type of the new index (heap, hash, b-tree, r-tree)
IN: ATTR_DESC **attr_info An array indicating the structure of the index attributes

**Description:**

To create an index

```
I_create(wa, indexname, indexset, mode, attr_info)
```

**Errors:**

If there is no such indexset, or attribute information is incorrect

**Examples:**

```
WA *wa /* work area definitions */
ATTR_DESC **attr_info; /* attribute information structure */  ...

WA_open ("nick r", "ccc",wa);
Fill_in_attr_info(attr_info);
I_create(wa, "objclass", "iueobs", "b-tree", attr_info);
```

**See Also:**

**Function Name:**

I_drop

**Purpose:**

To drop an index from an indexset

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: char *indexname The name of the source index
IN: char *indexset The name of source parent indexset

**Description:**

To drop an index

```
I_drop(indexname, indexset)
```

**Errors:**

If there is no such index, or such indexset.

**Examples:**

```
I_drop("objclass", "iueobs");
```

**See Also:**

**Function Name:**

I_insert

**Purpose:**

To insert a row into the current index of an indexset.

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

WA *wa A pointer to a work area
IN: char *buffer The buffer area holding the row to be inserted

**Description:**

To insert into an index:

```
I_insert(wa, buffer)
```

**Errors:**

If the current index is not open for modify. The type of the data is inconsistent with the defintion.

**Examples:**

```
+ WA *wa /* work area definitions */
TAG *tag1;
BUFFER *buffr;
 ...
WA_open ("nick r", "ccc",wa);
I_retrieve(wa, "objclass", "iueobs", 2, tag1);
I_pick (wa, tag1);
setup_row(buffr); /* sets the values of the columns */
I_first(wa);
I_next(wa);
I_insert(wa, buffer);
I_save(wa);
I_return(wa);
WA_close (wa);
```

**See Also:**

**Function Name:**

I_update

**Purpose:**

To update the current row of the current index of an indexset.

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: WA *wa A pointer to a work area
IN: char *row The new row to replace the existing one

**Description:**

To update current row of the current index:

```
I_update(wa, row)
```

**Errors:**

If the current index is not open for modify. The type of the data is inconsistent with the defintion.

**Examples:**

Suppose "objclass" is an index with two columns, "class" and "id", and we want to replace the second row of this index with the row (LWR,15)

```
WA *wa /* work area definitions */
 ...
TAG *tag1;
BUFFER *buffr1;

WA_open ("nick r", "ccc", wa);
I_retrieve(wa, "objclass", "iueobs", 1, tag1);
I_pick (wa, *tag1);
insert_to_buffer(buffr1,"LWR",15);
I_first(wa);
I_next(wa);
I_update(wa, buffr1);
I_save(wa);
I_return(wa, *tag1);
```

**See Also:**

**Function Name:**

I_move

**Purpose:**

To move an index from an indexset into another index

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: WA *wa A pointer to a work area
IN: char *indexname1 The name of the source index
IN: char *indexset1 The name of source parent indexset
IN: char *indexname2 The name of the destination index
IN: char *indexset2 The name of destination parent indexset

**Description:**

To move an index

```
I_move(wa, indexname1, indexset1, indexname2, indexset2)
```

**Errors:**

If there is no such source index, or such source indexset.

**Examples:**

```
WA *wa /* work area definitions */

WA_open ("nick r", "ccc",wa);
I_move(wa, "objclass", "iueobs", "objclass", "newiueobs");
```

**See Also:**

**Function Name:**

I_delete

**Purpose:**

To delete current row of the current index of an indexset.

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: WA *wa A pointer to a work area

**Description:**

To delete current row from the current index:

```
I_delete(wa)
```

**Errors:**

If the current index is not open for modify.

**Examples:**

```
WA *wa /* work area definitions */
TAG *tag1;
BUFFER *buffr;
VAR *classbuf;
 ...
WA_open ("nick r", "ccc",wa);
I_retrieve(wa, "objclass", "iueobs", 1, tag1);
I_crbind(wa, tag1, buffr);
I_ccbind(wa, tag1, "class", clasbuf);
I_pick (wa, tag1);
I_first(wa);
I_fetch(wa);
I_next(wa);
I_fetch(wa);
I_delete(wa);
I_dcbind(wa, tag1, "class", clasbuf);
```

**See Also:**

**Function Name:**

I_copy

**Purpose:**

To copy an index from an indexset into another index

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: WA *wa A pointer to a work area
IN: char *indexname1 The name of the source index
IN: char *indexset1 The name of source parent indexset
IN: char *indexname2 The name of the destination index
IN: char *indexset2 The name of destination parent indexset
IN: FORMAT *mode The type of the new index (heap, hash, b-tree, r-tree)

**Description:**

To copy an index

```
I_copy(wa, indexname1, indexset1, indexname2, indexset2, mode)
```

**Errors:**

If there is no such source index, or such source indexset.

**Examples:**

```
WA *wa /* work area definitions */
 ...
TAG *tag1, *tag2;

WA_open ("nick r", "ccc",wa);
I_copy(wa, "objclass", "iueobs", "objclass1", "newiueobs", "b-
tree");
```

**See Also:**

**Function Name:**

I_search

**Purpose:**

To search the current index of an indexset.

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: IN: WA *wa A pointer to a work area
IN: FILE *infile Name of file containing terms
IN/OUT: FILE *outfile Name of file containing pointers
IN/OUT: FILE *pairfile Name of file containing term-pointer pairs
IN/OUT: FILE *statfile Name of file containing statistics

**Description:**

To search an index:

```
I_search(wa, infile, outfile, pairfile, statfile)
```

**Errors:**

If there is no current index. If there is no input or output file.

**Examples:**

```
WA *wa /* work area definitions */
TAG *tag1;

WA_open ("nick r", "ccc",wa);
I_retrieve(wa, "objclass", "iueobs", 1, tag1);
I_pick (wa, *tag1);
I_search(wa, "windows", "output", "pairs", "stats");
```

For example, the contents of file "windows" may be:

```
99
24
```

Then, the contents of file "output" might be:

```
LWP2346
LWP2347
```

while the contents of file "pairs" will be:

```
99 LWP2346
24 LWP2347
```

The contents of file "stats" will be:

```
3 records found
```

**See Also:**

**Function Name:**

I_retrieve

**Purpose:**

To retrieve an index in an indexset.

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: WA *wa A pointer to a work area
IN: char *index Name of the index
IN: char *indexset Name of the indexset
IN: int mode mode of access e.g., read (0) or write (1). Default is read
IN/OUT: TAG *tag Optional tag name (the system generates a tag if none is specified)

**Description:**

To retrieve an index:

```
I_retrieve(wa, index, indexset, mode, tag)
```

**Errors:**

If there is no index. If there is no indexset. If the mode is not correct.

**Examples:**

```
+ WA *wa /* work area definitions */
 ...
TAG *tag1;
TAG *tag2;

WA_open ("nick r", "ccc",wa);
I_retrieve(wa, "objclass", "iueobs", 1, tag1);
I_retrieve(wa, "objclass", "rectss", 1, tag2);
```

**See Also:**

**Function Name:**

I_pick

**Purpose:**

To pick a index

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: WA *wa A pointer to a work area
IN: TAG tag The tag of the index

**Description:**

To pick an index

```
I_pick(wa, tag)
```

**Errors:**

If there is no tag of that name.

**Examples:**

```
WA *wa /* work area definitions */
 ...
TAG *tag1, *tag2;

WA_open ("nick r", "ccc",wa);
I_retrieve(wa, "objclass", "iueobs", 1, tag1);
I_retrieve(wa, "objclass", "rectss", 1, tag2);
I_pick (wa, *tag1);
I_first(wa);
process rows
I_pick (wa, *tag2);
I_first(wa);
process rows
```

**See Also:**

**Function Name:**

I_save

**Purpose:**

To save an index in an indexset.

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: WA *wa A pointer to a work area
IN: TAG tag The tag of the index

**Description:**

To save an index:

```
I_save(wa, tag)
```

**Errors:**

If there is no such tag

**Examples:**

```
+ WA *wa /* work area definitions */
 ...
TAG *tag1, *tag2;

WA_open ("nick r", "ccc",wa);
I_retrieve(wa, "objclass", "iueobs", 1, tag1);
I_retrieve(wa, "objclass", "rectss", 0, tag2);
update indexes
I_save(wa, *tag1);
I_save(wa, *tag2);
I_return(wa, *tag1);
I_return(wa, *tag2);
```

**See Also:**

**Function Name:**

I_return

**Purpose:**

To return an index in an indexset.

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: WA *wa A pointer to a work area
IN: TAG tag The tag of the index

**Description:**

To return an index:

```
I_return(wa, tag)
```

**Errors:**

If there is no such tag

**Examples:**

```
+ WA *wa /* work area definitions */
 ...
TAG *tag1, *tag2;

WA_open ("nick r", "ccc",wa);
I_retrieve(wa, "objclass", "iueobs", 1, tag1);
I_retrieve(wa, "objclass", "rectss", 0, tag2);
process indexes
I_return(wa, *tag1);
I_return(wa, *tag2);
```

**See Also:**

**Function Name:**

I_list

**Purpose:**

To list the retrieved indexes

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: WA *wa A pointer to a work area

**Description:**

To list all indexes

```
I_list(wa)
```

**Errors:**

None.

**Examples:**

```
WA *wa /* work area definitions */
 ...
TAG *tag1, *tag2;

WA_open ("nick r", "ccc",wa);
I_retrieve(wa, "objclass", "iueobs", 1, tag1);
I_retrieve(wa, "rects", "iueobs", 1, tag2);
I_list(wa);
```

The result is:

```
INDEX: iueobs/objclass TAG: I1
INDEX: iueobs/rects    TAG: I2
```

**See Also:**

**Function Name:**

I_first

**Purpose:**

To make the first row of the current index current.

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: WA *wa A pointer to a work area

**Description:**

To get the first row of the current index:

```
I_first(wa)
```

**Errors:**

**Examples:**

```
WA *wa /* work area definitions */
TAG *tag1;
BUFFER *buffr;
 ...
WA_open ("nick r", "ccc",wa);
I_retrieve(wa, "objclass", "iueobs", 1, tag1);
I_pick (wa, tag1);
I_first(wa);
buffr = I_fetch(wa);
print_row(buffr);
```

**See Also:**

**Function Name:**

I_next

**Purpose:**

To make the next row of the current index current.

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: WA *wa A pointer to a work area

**Description:**

To get the next row of the current index:

```
I_next(wa)
```

**Errors:**

**Examples:**

```
WA *wa /* work area definitions */
WA *wa /* work area definitions */
TAG *tag1;
BUFFER *buffr;
 ...
WA_open ("nick r", "ccc",wa);
I_retrieve(wa, "objclass", "iueobs", 1, tag1);
I_pick (wa, tag1);
I_first(wa);
buffr = I_fetch(wa);
print_row(buffr);
I_next(wa);
buffr = I_fetch(wa);
print_row(buffr);
```

**See Also:**

**Function Name:**

I_fetch

**Purpose:**

To get the current row of the current index.

**Function Module:**

uinms.a

**Function Type:**

pointer to character string (the buffer to hold the row)

**Parameters:**

IN: WA *wa A pointer to a work area

**Description:**

To get the current row of the current index:

```
I_fetch(wa)
```

**Errors:**

If there is no current row.

**Examples:**

```
WA *wa /* work area definitions */
TAG *tag1;
BUFFER *buffr;
 ...
WA_open ("nick r", "ccc",wa);
I_retrieve(wa, "objclass", "iueobs", 1, tag1);
I_pick (wa, tag1);
I_first(wa);
buffr = I_fetch(wa);
print_row(buffr);
I_next(wa);
buffr = I_fetch(wa);
print_row(buffr);
```

**See Also:**

**Function Name:**

I_last

**Purpose:**

To make the last row of the current index current.

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: WA *wa A pointer to a work area

**Description:**

To get the last row of the current index:

```
I_last(wa)
```

**Errors:**

**Examples:**

```
WA *wa /* work area definitions */
TAG *tag1;
BUFFER *buffr;
 ...
WA_open ("nick r", "ccc",wa);
I_retrieve(wa, "objclass", "iueobs", 1, tag1);
I_pick (wa, tag1);
I_last(wa);
buffr = I_fetch(wa);
print_row(buffr);
```

**See Also:**

**Function Name:**

I_previous

**Purpose:**

To make the previous row of the current index current.

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: WA *wa A pointer to a work area

**Description:**

To get the previous row of the current index:

```
I_previous(wa)
```

**Errors:**

**Examples:**

```
WA *wa /* work area definitions */
TAG *tag1;
BUFFER *buffr;
 ...
WA_open ("nick r", "ccc",wa);
I_retrieve(wa, "objclass", "iueobs", 1, tag1);
I_pick (wa, *tag1);
I_last(wa);
buffr = I_fetch(wa);
print_row(buffr);
I_previous(wa);
buffr = I_fetch(wa);
print_row(buffr);
```

**See Also:**

**Function Name:**

IB_build

**Purpose:**

To build a boolean for searching an index in an indexset.

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: WA *wa A pointer to a work area
IN: BOOL *boolean The boolean
IN/OUT: TAG *tag The tag of the boolean

**Description:**

To build a boolean for searching an index in an indexset:

```
IB_build (wa, boolean, tag);
```

**Errors:**

If there is an incorrectly specified boolean.

**Examples:**

```
WA *wa /* work area definitions */
 ...
TAG *tag1;
BUFFER *buffr1;

TAG *tag2;
BUFFER *buffr2;

WA_open ("nick r", "ccc",wa);
I_retrieve(wa, "objclass", "iueobs", 1, tag1);
I_crbind(wa, tag1, buffr1);
```

To search for the first row of a b-tree index with named objclass in indexset iueobs satisfying the condition objclass = 99

```
I_pick (wa, "i1");
strcpy(log_txt_str, "objclass=99");
IB_build (wa, log_txt_str, "b1");
IB_pick(wa, "b1");
I_first(wa);
I_fetch(wa);
```

**See Also:**

**Function Name:**

IB_list

**Purpose:**

To list the booleans.

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: WA *wa A pointer to a work area

**Description:**

To list all booleans:

```
IB_list(wa)
```

**Errors:**

None.

**Examples:**

```
WA *wa /* work area definitions */
 ...
TAG *tag1, *b1, *b2;
BUFFER *log_txt_str;

WA_open ("nick r", "ccc",wa);
I_retrieve(wa, "objclass", "iueobs", 1, tag1);
I_pick (wa, tag1);
strcpy(log_txt_str, "objclass=99");
IB_build (wa, log_txt_str, b1);
strcpy(log_txt_str, "objclass=101");
IB_build (wa, log_txt_str, b2);
IB_list(wa);
```

The result is:

```
BOOLEAN: objclass=99 TAG: B1
BOOLEAN: objclass=101 TAG: B2
```

**See Also:**

**Function Name:**

IB_pick

**Purpose:**

To pick a boolean.

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: WA *wa A pointer to a work area
IN: TAG tag The tag of the boolean

**Description:**

To pick a boolean:

```
IB_pick(wa, tag)
```

**Errors:**

If there is no tag of that name.

**Examples:**

```
WA *wa /* work area definitions */
 ...
TAG *tag1, *b1;
BOOL *log_txt_str;

WA_open ("nick r", "ccc",wa);
I_retrieve(wa, "objclass", "iueobs", 1, tag1);
I_pick (wa, *tag1);
strcpy(log_txt_str, "objclass=99");
IB_build (wa, log_txt_str, b1);
IB_pick(wa, *b1);
I_first(wa);
 ....
```

**See Also:**

**Function Name:**

IB_modify

**Purpose:**

To modify a boolean for searching an index in an indexset.

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: WA *wa A pointer to a work area
IN: BOOL *boolean The boolean expression

**Description:**

To modify a boolean

```
IB_modify (wa, boolean);
```

**Errors:**

If there is an incorrectly specified boolean.

**Examples:**

```
WA *wa /* work area definitions */
 ...
TAG *tag1;
BOOL *boolean;
char *buffer;

WA_open ("nick r", "ccc", wa);
I_retrieve(wa, "objclass", "iueobs", 1, tag1);
I_pick (wa, tag1);
strcpy(boolean, "objclass=99");
IB_build (wa, boolean, b1);
IB_pick(wa, *b1);
I_first(wa);
buffer = I_fetch(wa);
print_row(buffer);
strcpy(boolean, "objclass=101");
IB_modify(wa, boolean);
I_first(wa);
buffer = I_fetch(wa);
print_row(buffer);
WA_close (wa);
```

**See Also:**

**Function Name:**

IB_drop

**Purpose:**

To drop a boolean.

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: WA *wa A pointer to a work area
IN: TAG tag The tag of the boolean

**Description:**

To drop a boolean:

```
IB_drop(wa, tag);
```

**Errors:**

If there is no such tag

**Examples:**

```
WA *wa /* work area definitions */
 ...
TAG *tag1, *b1;
BUFFER *buffr;

WA_open ("nick r", "ccc", wa);
I_retrieve(wa, "objclass", "iueobs", 1, tag1);
I_crbind(wa, tag1, buffr);
I_pick (wa, tag1);
strcpy(log_txt_str, "objclass=99");
IB_build (wa, log_txt_str, b1);
IB_pick(wa, *b1);
I_first(wa);
I_fetch(wa);
print_row(buffr);
IB_drop((wa, *b1);
I_drbind(wa, tag1, buffr);
```

**See Also:**

**Function Name:**

I_ccbind

**Purpose:**

To bind a column to a buffer area

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: WA *wa A pointer to a work area
IN: char *column The column to bind
IN: char *var The buffer area (variable)
IN: char* var_type The type of the variable to be used
IN: char *var_len The length of the variable to be used

**Description:**

To bind a column to a buffer area

```
I_ccbind(wa, column, var, var_type, var_len)
```

**Errors:**

If the column does not exist, or space has not been allocated for the buffer area

**Examples:**

```
WA *wa /* work area definitions */
TAG *tag1;
BUFFER *buffr;
VAR *classbuf;
 ...
WA_open ("nick r", "ccc",wa);
I_retrieve(wa, "objclass", "iueobs", 1, tag1);
I_pick (wa, tag1);
I_crbind(wa, buffr);
I_ccbind(wa, "class", clasbuf, "string", 20);
I_first(wa);
I_fetch(wa);
I_next(wa);
I_fetch(wa);
I_delete(wa);
I_dcbind(wa, "class", classbuf);
```

**See Also:**

**Function Name:**

I_dcbind

**Purpose:**

To drop a binding of a column

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: WA *wa A pointer to a work area
IN: char *column The column to bind
IN: char *buffer The buffer area

**Description:**

To drop a binding of a column

```
I_dcbind(wa, column, buffer)
```

**Errors:**

If the column does not exist or the binding was never established

**Examples:**

```
WA *wa /* work area definitions */
TAG *tag1;
BUFFER *buffr;
VAR *classbuf;
 ...
WA_open ("nick r", "ccc",wa);
I_retrieve(wa, "objclass", "iueobs", 1, tag1);
I_pick (wa, tag1);
I_crbind(wa, buffr);
I_ccbind(wa, "class", clasbuf);
I_first(wa);
I_fetch(wa);
I_next(wa);
I_fetch(wa);
I_delete(wa);
I_dcbind(wa, "class", classbuf);
```

**See Also:**

**Function Name:**

I_crbind

**Purpose:**

To bind a row to a buffer area

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: WA *wa A pointer to a work area
IN/OUT: char *buffer The buffer area

**Description:**

To bind a row to a buffer area

```
I_crbind(wa, buffer)
```

**Errors:**

If space has not been allocated for the buffer area

**Examples:**

```
WA *wa /* work area definitions */
TAG *tag1;
BUFFER *buffr;
VAR *classbuf;
 ...
WA_open ("nick r", "ccc",wa);
I_retrieve(wa, "objclass", "iueobs", 1, tag1);
I_pick (wa, tag1);
I_crbind(wa, buffr);
I_first(wa);
I_fetch(wa);
print_row(buffr);
I_next(wa);
I_fetch(wa);
print_row(buffr);
I_delete(wa);
I_drbind(wa, buffr);
```

**See Also:**

**Function Name:**

I_drbind

**Purpose:**

To drop the binding of a row

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: WA *wa A pointer to a work area
IN: char *buffer The buffer area

**Description:**

To drop the binding of a row

```
I_drbind(wa, buffer)
```

**Errors:**

If the binding was never established

**Examples:**

```
WA *wa /* work area definitions */
TAG *tag1;
BUFFER *buffr;
VAR *classbuf;
 ...
WA_open ("nick r", "ccc",wa);
I_pick (wa, tag1);
I_retrieve(wa, "objclass", "iueobs", 1, tag1);
I_crbind(wa, buffr);
I_first(wa);
I_fetch(wa);
print_row(buffr);
I_next(wa);
I_fetch(wa);
print_row(buffr);
I_delete(wa);
I_drbind(wa, buffr);
```

**See Also:**

**Function Name:**

WA_open

**Purpose:**

To initialize a work area.

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: USER *user The name of the user
IN: PASS *pass The password of the user
OUT: WA *wa A pointer to a work area

**Description:**

To initialize a work area.

```
WA_open (user, pass, wa);
```

**Errors:**

If the user, or password is incorrect.

**Examples:**

```
WA *wa /* work area definitions */
 ...
WA_open ("nick r", "ccc", wa);
 ...
WA_close (wa);
```

**See Also:**

**Function Name:**

WA_close

**Purpose:**

To close a work area.

**Function Module:**

uinms.a

**Function Type:**

integer. Returns OK or an error code

**Parameters:**

IN: WA *wa A pointer to a work area (ret)

**Description:**

To close a work area.

```
WA_close (wa);
```

**Errors:**

If the work area pointer is incorrect.

**Examples:**

```
WA *wa /* work area definitions */
 ...
WA_open ("nick r", "ccc", wa);
 ...
WA_close (wa);
```

**See Also:**

# Command Language Interface

**Command Name:**

create indexset

**Purpose:**

To create an indexset.

**Command Module:**

uinms.exe

**Synopsis:**

create indexset <indexset>

**Parameters:**

indexset- name of parent indexset

**Description:**

To create an indexset

```
create indexset <indexset>
```

**Errors:**

If there is an existing indexset with the same name

**Examples:**

create indexset iueobs

**See Also:**

**Command Name:**

delete indexset

**Purpose:**

To delete an indexset.

**Command Module:**

uinms.exe

**Synopsis:**

delete indexset <indexset>

**Parameters:**

indexset- name of parent indexset

**Description:**

To delete an indexset

```
delete indexset <indexset>
```

**Errors:**

If there is no existing indexset with name given

**Examples:**

delete indexset iueobs

**See Also:**

**Command Name:**

create

**Purpose:**

To create an index in an indexset.

**Command Module:**

uinms.exe

**Synopsis:**

create index <indexname> <indexset> <mode> <definition_file>

**Parameters:**

indexname- name of the index
indexset- name of parent indexset
mode- mode of access (e.g., heap, hash, b-tree, r-tree) Default is heap
definition_file- the file containing the schema of the index

**Description:**

To create an index:

```
create index <indexname> <indexset> <mode> <definition_file>
```

**Errors:**

If there is no indexset. If there is no correct mode. etc.

**Examples:**

To create a b-tree index with name "objclass" in indexset "iueobs" with definition file "schema"

```
create index objclass iueobs b-tree schema
```

Assuming that the two fields in the index are "name" and "id", the file "schema" will have the following information

```
name string  30
id    int     4
```

If we would have liked to create an r-tree index on a set of rectangles, the file "schema" would contain

```
attra int   4
attrb int   4
attrc int   4
attrd int   4
id    int   4
```

**See Also:**

## Command Name:

drop

## Purpose:

To drop an index in an indexset.

## Command Module:

uinms.exe

## Synopsis:

drop index <indexname> <indexset>

## Parameters:

indexname- name of the index
indexset- name of parent indexset

## Description:

To drop an index:

```
drop index <indexname> <indexset>
```

## Errors:

If there is no such index, or such indexset.

## Examples:

To drop an index with name "objclass" in indexset "iueobs"

```
drop index objclass iueobs
```

## See Also:

**Command Name:**

insert

**Purpose:**

To insert a row into the current index of an indexset.

**Command Module:**

uinms.exe

**Synopsis:**

insert into index <row>

**Parameters:**

row- the row to be inserted

**Description:**

To insert into an index:

```
insert into index <row>
```

**Errors:**

If the current index is not open for modify. The type of the data is inconsistent with the defintion.

be inserted using the symbol "|"

**Examples:**

```
retrieve index objclass iueobs modify i1
retrieve index rects iueobs modify i2
```

To insert into a b-tree index with name objclass in indexset iueobs with format:

```
objclass integer
cam string 9
```

we do the following:

```
pick i1
insert into index objclass|99|cam|LWP2344
```

To insert into an r-tree index with name rects in indexset iueobs with format:

```
ra1 integer 4
dec1 integer 4
ra2 integer 4
dec2 integer 4
cam string 9
```

we do the following:

```
pick i2
insert into index ra1|2344|dec1|37384|ra2|39494|dec2|23399|cam|LWP2346
```

**See Also:**

**Command Name:**

update

**Purpose:**

To update current row of the current index of an indexset.

**Command Module:**

uinms.exe

**Synopsis:**

update index <row>

**Parameters:**

row- the row to be updateed

**Description:**

To update current row of the current index:

```
update index <row>
```

**Errors:**

If the current index is not open for modify. The type of the data is inconsistent with the defintion.

**Examples:**

```
retrieve index objclass iueobs modify i1
retrieve index rects iueobs modify i2
```

To update the first row of a b-tree index "objclass" in "iueobs" with format :

```
objclass integer
camseq string 9
```

we do the following:

```
pick i1
first in index
fetch from index
update index objclass|99|camseq|LWP2345
```

To update the first row of an r-tree index "objclass" in "iueobs" with format

```
ra1 integer 4          dec1 integer 4
ra2 integer 4          dec2 integer 4
camseq string 9
```

we do the following:

```
pick i2
first in index
fetch from index
update index ra1|2344|dec1|37384|ra2|39494|dec2|23399|camseq|LWP2348
```

**See Also:**

insert

### Command Name:

move

### Purpose:

To move an index from an indexset into another index

### Command Module:

uinms.exe

### Synopsis:

move index <indexname1> <indexset1> <indexname2> <indexset2>

### Parameters:

indexname1- name of the source index
indexset1- name of source parent indexset
indexname2- name of the destination index
indexset2- name of destination parent indexset

### Description:

To move an index:

```
move index <indexname1> <indexset1> <indexname2> <indexset2>
```

### Errors:

If there is no such source index, or such source indexset. The new index retains the structure of the source index.

### Examples:

To move an index with name "objclass" in indexset "iueobs" into index "objclass2" in indexset "oldiue"

```
move index objclass iueobs objclass2 oldiue
```

### See Also:

**Command Name:**

delete

**Purpose:**

To delete current row of the current index of an indexset.

**Command Module:**

uinms.exe

**Synopsis:**

delete from index

**Parameters:**

none

**Description:**

To delete current row from the current index:

```
delete from index
```

**Errors:**

If the current index is not open for modify.

**Examples:**

```
retrieve index objclass iueobs modify i1
retrieve index rects iueobs modify i2
```

To delete the first row of a b-tree index "objclass" in "iueobs" with format :

```
objclass integer
camseq string 9
```

we do the following:

```
pick i1
first in index
fetch from index
delete from index
```

To delete the second row of an r-tree index "rects" in "iueobs" with format:

```
ra1 integer 4          dec1 integer 4
ra2 integer 4          dec2 integer 4
camseq string 9
```

we do the following:

```
pick i2
first in index
fetch from index
next in index
fetch from index
delete from index
```

**See Also:**

**Command Name:**

copy

**Purpose:**

To copy an index from an indexset into another index

**Command Module:**

uinms.exe

**Synopsis:**

copy index <indexname1> <indexset1> <indexname2> <indexset2> <mode>

**Parameters:**

indexname1- name of the source index
indexset1- name of source parent indexset
indexname2- name of the destination index
indexset2- name of destination parent indexset
mode- type of the new index (heap, hash, b-tree, r-tree)

**Description:**

To copy an index:

```
copy index <indexname1> <indexset1> <indexname2> <indexset2> <mode>
```

**Errors:**

If there is no such source index, or such source indexset.

**Examples:**

To copy an index with name "objclass" in indexset "iueobs" into another b-tree index "objclass2" in indexset "oldiue"

```
copy index objclass iueobs objclass2 oldiue b-tree
```

**See Also:**

**Command Name:**

search

**Purpose:**

To search the current index of an indexset.

**Command Module:**

uinms.exe

**Synopsis:**

search index <infile> <outfile> <statfile>

**Parameters:**

infile- name of file containing terms
outfile- name of file containing pointers
pairfile- name of file containing term-pointer pairs
statfile- name of file containing statistics

**Description:**

To search an index:

```
search index <infile> <outfile> [<pairfile>] [<statfile>]
```

**Errors:**

If there is no index. If there is no indexset. If there is no input file.

**Examples:**

```
retrieve index objclass iueobs read-only i1
retrieve index rects iueobs read-only i2
```

To search a b-tree index with name objclass in indexset iueobs where the input file is "windows" and the output file is "observations", pairsfile is "pairs" and the statfile is "stats"

```
pick i1
search index windows observations pairs stats
```

The contents of file "windows" will be:

```
99
24
38
```

The contents of file "observations" will be:

```
LWP2346
LWP2346
LWP2347
```

The contents of file "pairs" will be:

```
99 LWP2346
24 LWP2346
24 LWP2347
```

The contents of file "stats" will be:

```
3 records found
```

To search n r-tree index with name objclass in indexset iueobs where the input file

is "windows" and the output file is "rects.dat", pairsfile is "pairs" and the statfile is "stats"

```
pick i2
search index rects iueobs windows rects.dat pairs stats
```

the contents of file "windows" will be:

```
2344 37384 39494 23399
5344 47384 33394 83399
5344 47384 33394 83399
```

the contents of file "rects.dat" will be:

```
LWP2346
LWP2347
LWP2349
```

the contents of file "pairs" will be:

```
2344 37384 39494 23399  LWP2346
5344 47384 33394 83399  LWP2347
5344 47384 33394 83399  LWP2347
```

The contents of file "stats" will be:

```
3 records found
```

**See Also:**

## Command Name:

retrieve

## Purpose:

To retrieve an index in an indexset.

## Command Module:

uinms.exe

## Synopsis:

retrieve index <indexname> <indexset> <mode> [<tag>]

## Parameters:

indexname- name of the index
indexset- name of parent indexset
mode- mode of access (e.g., read or write) Default is read
tag- optional tag name (the system generates a tage if none is specified)

## Description:

To retrieve an index:

```
retrieve index <indexname> <indexset> <mode> [<tag>]
```

## Errors:

If there is no index. If there is no indexset. If there is no correct mode. etc.

## Examples:

To retrieve a b-tree index with name objclass in indexset iueobs where the mode is read:

```
retrieve index objclass iueobs read-only
```

To retrieve a hash index with name objclass in indexset iueobs where the mode is write :

```
retrieve index objclass iueobs modify t3
```

To retrieve an r-tree index with name rects in indexset iueobs where the mode is read:

```
retrieve index rects iueobs read-only
```

## See Also:

**Command Name:**

pick

**Purpose:**

To pick an index in an indexset and make it current.

**Command Module:**

uinms.exe

**Synopsis:**

pick index <tag>

**Parameters:**

tag- tag name

**Description:**

To pick an index:

```
pick index <tag>
```

**Errors:**

If there is no tag of that name.

**Examples:**

To pick an index t1:

```
pick index t1
```

**See Also:**

**Command Name:**

save

**Purpose:**

To save an index in an indexset.

**Command Module:**

uinms.exe

**Synopsis:**

save index <tag>

**Parameters:**

tag- tag name

**Description:**

To save an index:

```
save index <tag>
```

**Errors:**

If there is no tag of that name.

**Examples:**

To save a b-tree index t1 with name objclass in indexset iueobs where the mode is modify:

```
save index t1
```

To save a hash index t2 with name objclass in indexset iueobs where the mode is modify:

```
save index t2
```

To save an r-tree index t3 with name rects in indexset iueobs where the mode is modify:

```
save index t3
```

**See Also:**

**Command Name:**

return

**Purpose:**

To return an index in an indexset.

**Command Module:**

uinms.exe

**Synopsis:**

return index <tag>

**Parameters:**

tag- tag name

**Description:**

To return an index:

```
return index <tag>
```

**Errors:**

If there is no tag of that name.

**Examples:**

To return a b-tree index t1 with name objclass in indexset iueobs where the mode is read:

```
return index t1
```

To return a hash index t2 with name objclass in indexset iueobs where the mode is write :

```
return index t2
```

To return an r-tree index t3 with name rects in indexset iueobs where the mode is read:

```
return index t3
```

**See Also:**

**Command Name:**

list

**Purpose:**

To list the retrieved indexes.

**Command Module:**

uinms.exe

**Synopsis:**

list index

**Parameters:**

none

**Description:**

To list all retreived indexes:

```
list index
```

**Errors:**

If there is no tag of that name.

**Examples:**

```
retrieve index objclass iueobs modify i1
retrieve index rects iueobs modify i2
```

To list all indexes:

```
list index
```

The result is:

```
INDEX: iueobs/objclass TAG: i1
INDEX: iueobs/rects    TAG: i2
```

**See Also:**

**Command Name:**

first

**Purpose:**

To make the first row of the current index current.

**Command Module:**

uinms.exe

**Synopsis:**

first in index

**Parameters:**

none

**Description:**

To get the first row of the current index:

```
first in index
```

**Errors:**

If no index has been picked as current.

**Examples:**

```
retrieve index objclass iueobs modify i1
retrieve index rects iueobs modify i2
```

To retrieve first row of index i1:

```
pick i1
first in index
fetch from index
```

To retrieve first row of index i2:

```
pick i2
first in index
fetch from index
```

**See Also:**

**Command Name:**

next

**Purpose:**

To make the next row of the current index current.

**Command Module:**

uinms.exe

**Synopsis:**

next in index

**Parameters:**

none

**Description:**

To get the next row of the current index:

```
next in index
```

**Errors:**

If no index has been picked as current.

**Examples:**

```
retrieve index objclass iueobs modify i1
retrieve index rects iueobs modify i2
```

To retrieve next row of index i1:

```
pick i1
first in index
next in index
fetch from index
```

To retrieve next row of index i2:

```
pick i2
first in index
next in index
fetch from index
```

**See Also:**

**Command Name:**

fetch

**Purpose:**

To display the current row of the current index.

**Command Module:**

uinms.exe

**Synopsis:**

fetch from index

**Parameters:**

none

**Description:**

To display the current row of the current index:

```
fetch from index
```

**Errors:**

If there is no current row.

**Examples:**

```
retrieve index objclass iueobs modify i1
retrieve index rects iueobs modify i2
```

To fetch last row of index i1:

```
pick i1
last in index
fetch from index
```

The last row is:

```
99   "LWP2345"
```

To fetch first row of index i2:

```
pick i2
first in index
fetch from index
```

The first row is:

```
2344 37384 39494 23399 "LWP2348"
```

**See Also:**

**Command Name:**

last

**Purpose:**

To make the last row of the current index current.

**Command Module:**

uinms.exe

**Synopsis:**

last in index

**Parameters:**

none

**Description:**

To get the last row of the current index:

```
last in index
```

**Errors:**

If no index has been picked as current.

**Examples:**

```
retrieve index objclass iueobs modify i1
retrieve index rects iueobs modify i2
```

To retrieve last row of index i1:

```
pick i1
first in index
last in index
fetch from index
```

To retrieve last row of index i2:

```
pick i2
first in index
last in index
fetch from index
```

**See Also:**

**Command Name:**

previous

**Purpose:**

To make the previous row of the current index current.

**Command Module:**

uinms.exe

**Synopsis:**

previous in index

**Parameters:**

none

**Description:**

To get the previous row of the current index:

```
previous in index
```

**Errors:**

If no index has been picked as current.

**Examples:**

```
retrieve index objclass iueobs modify i1
retrieve index rects iueobs modify i2
```

To retrieve previous row of index i1:

```
pick i1
last in index
previous in index
fetch from index
```

To retrieve previous row of index i2:

```
pick i2
last in index
previous in index
fetch from index
```

**See Also:**

**Command Name:**

build boolean

**Purpose:**

To build a boolean for searching an index in an indexset.

**Command Module:**

uinms.exe

**Synopsis:**

build boolean <expression> [<tag>]

**Parameters:**

expression - boolean condition
tag- optional tag name (the system generates a tag if none is specified)

**Description:**

To build a boolean for searching an index in an indexset:

```
build boolean <expression> [<tag>]
```

**Errors:**

If there is an incorrectly specified boolean.

**Examples:**

```
retrieve index objclass iueobs modify i1
retrieve index rects iueobs modify i2
```

To search for the first row of a b-tree index with name objclass in indexset iueobs satisfying the condition objclass = 99:

```
pick index i1
build boolean objclass=99 b1
pick boolean b1
first in index
fetch from index
```

To search for the first row of an r-tree index with name rects in indexset iueobs satisfying the condition (ra1,dec1,ra2,dec2) overlaps (23,3456,85754,5599):

```
pick index i2
build boolean (ra1,dec1,ra2,dec2) OV (23,3456,85754,5599) b1
pick boolean b1
first in index
fetch from index
```

**See Also:**

**Command Name:**

list boolean

**Purpose:**

To list the booleans.

**Command Module:**

uinms.exe

**Synopsis:**

list boolean

**Parameters:**

none

**Description:**

To list all booleans:

```
list boolean
```

**Errors:**

None.

**Examples:**

```
retrieve index objclass iueobs modify i1
retrieve index rects iueobs modify i2
```

To search for the first row of a b-tree index with name objclass in indexset iueobs satisfying the condition objclass = 99:

```
build boolean objclass=99 b1
build boolean (ra1,dec1,ra2,dec2) OV (23,3456,85754,5599) b1
```

To list all boolean:

```
list boolean
```

The result is:

```
BOOLEAN: objclass=99 TAG: b1
BOOLEAN: (ra1,dec1,ra2,dec2) OV (23,3456,85754,5599) TAG: b2
```

**See Also:**

**Command Name:**

pick boolean

**Purpose:**

To pick a boolean.

**Command Module:**

uinms.exe

**Synopsis:**

pick boolean <tag>

**Parameters:**

tag- tag name

**Description:**

To pick an boolean:

```
pick boolean <tag>
```

**Errors:**

If there is no tag of that name.

**Examples:**

To pick an boolean b1:

```
pick boolean b1
```

**See Also:**

**Command Name:**

modify boolean

**Purpose:**

To modify current boolean.

**Command Module:**

uinms.exe

**Synopsis:**

modify boolean <boolean>

**Parameters:**

boolean- the new boolean

**Description:**

To modify current boolean :

```
modify boolean <boolean>
```

**Errors:**

If the boolean is syntactically incorrect.

**Examples:**

```
retrieve index objclass iueobs modify i1
retrieve index rects iueobs modify i2
```

To search for the first row of a b-tree index with name objclass in indexset iueobs satisfying the condition objclass = 99:

```
build boolean objclass=99 b1
build boolean (ra1,dec1,ra2,dec2) OV (23,3456,85754,5599) b1
```

To modify boolean b1:

```
pick boolean b1
modify boolean objclass=24
```

To modify boolean b2:

```
pick boolean b2
modify boolean (ra1,dec1,ra2,dec2) OV (23,3456,85754,5598)
```

**See Also:**

**Command Name:**

drop boolean

**Purpose:**

To drop a boolean.

**Command Module:**

uinms.exe

**Synopsis:**

drop boolean <tag>

**Parameters:**

tag- tag name

**Description:**

To drop a boolean:

```
drop boolean <tag>
```

**Errors:**

If there is no tag of that name.

**Examples:**

To drop boolean b1:

```
drop boolean b1
```

**See Also:**

**Command Name:**

create indexset

**Purpose:**

To create an indexset.

**Command Module:**

uinms.exe

**Synopsis:**

create indexset <indexset>

**Parameters:**

indexset- name of parent indexset

**Description:**

To create an indexset

```
create indexset <indexset>
```

**Errors:**

If there is an existing indexset with the same name

**Examples:**

create indexset iueobs

**See Also:**

**Command Name:**

delete indexset

**Purpose:**

To delete an indexset.

**Command Module:**

uinms.exe

**Synopsis:**

delete indexset <indexset>

**Parameters:**

indexset- name of parent indexset

**Description:**

To delete an indexset

```
delete indexset <indexset>
```

**Errors:**

If there is no existing indexset with name given

**Examples:**

delete indexset iueobs

**See Also:**

# UIMS TEST SUITE

**Steve Kelley**
**Nick Roussopoulos**
**Timos Sellis**

Advanced Communication Technology Inc.
1209 Goth Lane
Silver Spring, Maryland 20905

**Final Report**

**SBIR Phase II Contract Number NAS5-30628**

Prepared for
Goddard Space Flight Center
Greenbelt, Maryland 20771

**November 10, 1992**

# README FILE

This directory tree contains the source code, libraries, executable, script, data and test directorys for the NASA sponsored UIMS system.

## Directory List and Contents:

README - this file

Source Code Directories:

- include - header files (.h) which define UIMS internal structures
- bTREE - source and object code for "B-Tree" index type
- bind - soruce code for binding indexes to names
- bool - source and object code which deal with boolean expressions
- index - source and objsect code common to all index types
- minirel - source and object code for basic file operations
- rtindex - r-tree index source and object clode (not available)
- sec - source and object code for B-Tree creation
- sec_aux - utility source and object code for B-Trees
- select - source and object code for index selection
- tools - source and object code for UIMS utilities

Library Directory:

- lib - object code libraries (made from the source directories)

Executable Directory:

- bin - where the "index" executable is created/run from

Test Directories:

- script - contains test scripts - command execution files
- data - contains test data used by the scripts
- db - (empty) directory where test may scripts be run

## To Make a new version of the system:

Change directory (cd) to each of the source code directories (excluding "include" and saving "index" for last) an type:

```
make all
```

For each directory excpting "index", this will compile the source code into object code and create an objdect code library in "lib". Typing this in "index" will result in compiling the source code thare and then link it with the libraries to create the "index" executable in "bin".

## Testing the System

If you wish to test the system, there are script files in UIMS/script that act as input to an interactive program demonstrating the system.

Change to the "db" directory then execute the following command scripts.
They will completely test all functioning user level routines.

```
../bin/index < ../script/input.create.load.unload.help > test1
../bin/index < ../script/input.copy.move > test2
../bin/index < ../script/input.retrieve.navigate.return > test3
../bin/index < ../script/input.boolean.navigate > test4
../bin/index < ../script/input.modify > test5
../bin/index < ../script/input.drop > test6
```

All the 'test' files will show the error codes/return values for each command in the input file. There are some commands that are intended to give error codes back (e.g. trying to update an index that was retrieved for read only, requesting the previous tuple when currently at the beginning of the index, etc.).

## TEST SUIT

The test suite consiste of 6 scripts. Below are the scripts and the output of each of these tests.

### SCRIPT 1

```
input.create.load.unload.help
sarah
passwd
create indexset sarahiset
create indexset aliset
create indexset jeniset
create indexset joiset
create index indexhp sarahiset heap ../data/heap.def
create index indexhs sarahiset hash ../data/hash.def
create index indexbt sarahiset btree ../data/btree.def
create index indexrt sarahiset rtree ../data/rtree.def
load index indexhp sarahiset ../data/sarahiset.data
load index indexhs sarahiset ../data/aliset.data
load index indexbt sarahiset ../data/jeniset.data
load index indexrt sarahiset ../data/rtree.data
unload index indexhp sarahiset indexhp.unload
unload index indexhs sarahiset indexhs.unload
unload index indexbt sarahiset indexbt.unload
unload index indexrt sarahiset indexrt.unload
help index indexhp sarahiset indexhp.help
help index indexhs sarahiset indexhs.help
help index indexbt sarahiset indexbt.help
help index indexrt sarahiset indexrt.help
quit
```

## SCRIPT 2

```
input.copy.move
sarah
passwd
copy index indexhp sarahiset indexhp aliset
copy index indexhs sarahiset indexhs aliset
copy index indexbt sarahiset indexbt aliset
copy index indexrt sarahiset indexrt aliset
copy index indexhp sarahiset indexhs1 jeniset hash
copy index indexhp sarahiset indexbt1 jeniset btree
copy index indexhs sarahiset indexhp1 jeniset heap
copy index indexhs sarahiset indexbt2 jeniset btree
copy index indexbt sarahiset indexhp2 jeniset heap
copy index indexbt sarahiset indexhs2 jeniset hash
move index indexhp sarahiset indexhp joiset
move index indexhs sarahiset indexhs joiset
move index indexbt sarahiset indexbt joiset
move index indexrt sarahiset indexrt joiset
quit
```

SCRIPT 3

```
input.retrieve.navigate.return
sarah
passwd
retrieve index indexhp joiset read_only ihp
list index
pick index ihp
first in index
fetch from index
previous in index
first in index
next in index
fetch from index
last in index
fetch from index
next in index
last in index
previous in index
fetch from index
return index ihp
retrieve index indexhs joiset read_only ihs
pick index ihs
first in index
fetch from index
previous in index
first in index
next in index
fetch from index
last in index
fetch from index
next in index
last in index
previous in index
fetch from index
return index ihs
retrieve index indexbt joiset read_only ibt
pick index ibt
first in index
fetch from index
previous in index
first in index
next in index
fetch from index
last in index
fetch from index
next in index
last in index
previous in index
fetch from index
return index ibt
retrieve index indexrt joiset read_onlu irt
pick index irt
batch search index ../data/rtree.search rtree-id.out rtree-tup.out
```

```
return index irt
quit
```

## SCRIPT 4

```
input.boolean.navigate
sarah
passwd
retrieve index indexhp joiset read_only ihp
list index
pick index ihp
build boolean attra < "barbara" ib1
pick boolean ib1
first in index
fetch from index
previous in index
first in index
next in index
fetch from index
last in index
fetch from index
next in index
last in index
previous in index
fetch from index
return index ihp
retrieve index indexhs joiset read_only ihs
pick index ihs
pick boolean ib1
first in index
fetch from index
previous in index
first in index
next in index
fetch from index
last in index
fetch from index
next in index
last in index
previous in index
fetch from index
return index ihs
retrieve index indexbt joiset read_only ibt
pick index ibt
pick boolean ib1
first in index
fetch from index
previous in index
first in index
next in index
fetch from index
last in index
fetch from index
next in index
last in index
previous in index
fetch from index
```

```
return index ibt
drop boolean ib1
quit
```

SCRIPT 5

```
input.modify
sarah
passwd
retrieve index indexhp joiset modify ihp
list index
pick index ihp
last in index
fetch from index
update index attra|timos sellis|id|3030
last in index
fetch from index
delete from index
last in index
fetch from index
insert into index attra|nick roussopoulous|id|9090
last in index
fetch from index
save index ihp
return index ihp
retrieve index indexhs joiset modify ihs
pick index ihs
last in index
fetch from index
update index attra|richard wallace|id|3030
last in index
fetch from index
delete from index
last in index
fetch from index
insert into index attra|nick roussopoulous|id|9090
last in index
fetch from index
save index ihs
return index ihs
retrieve index indexbt joiset modify ibt
pick index ibt
last in index
fetch from index
update index attra|timos sellis|id|3030
last in index
fetch from index
delete from index
last in index
fetch from index
insert into index attra|nick roussopoulous|id|9090
last in index
fetch from index
save index ibt
return index ibt
retrieve index indexhp joiset read_only ihp
list index
pick index ihp
```

```
last in index
fetch from index
update index attra|timos sellis|id|3030
last in index
fetch from index
delete from index
last in index
fetch from index
insert into index attra|nick roussopoulous|id|9090
last in index
fetch from index
save index ihp
return index ihp
retrieve index indexhs joiset read_only ihs
pick index ihs
last in index
fetch from index
update index attra|timos sellis|id|3030
last in index
fetch from index
delete from index
last in index
fetch from index
insert into index attra|nick roussopoulous|id|9090
last in index
fetch from index
save index ihs
return index ihs
retrieve index indexbt joiset read_only ibt
pick index ibt
last in index
fetch from index
update index attra|timos sellis|id|3030
last in index
fetch from index
delete from index
last in index
fetch from index
insert into index attra|nick roussopoulous|id|9090
last in index
fetch from index
save index ibt
return index ibt
quit
```

## SCRIPT 6

```
input.drop
sarah
passwd
drop index indexhp aliset
drop index indexhs aliset
drop index indexbt aliset
drop index indexhp joiset
drop index indexhs joiset
drop index indexbt joiset
drop index indexhp1 jeniset
drop index indexhp2 jeniset
drop index indexhs1 jeniset
drop index indexhs2 jeniset
drop index indexbt1 jeniset
drop index indexbt2 jeniset
delete indexset sarahiset
delete indexset aliset
delete indexset jeniset
delete indexset joiset
quit
```

```
                                    TEST 1
Please enter your username: Please enter your password:
Need to validate user and password.
COMMANDS AND THEIR SYNTAX:

create index  <indexname> <indexset> <format> <infile>
drop index    <indexname> <indexset>
copy index    <from-index> <from-indexset> <to-index> <to-indexset> [<format>]
move index    <from-index> <from-indexset> <to-index> <to-indexset>
help index    <indexname> <indexset>
load index    <indexname> <indexset> <infile>
unload index  <indexname> <indexset> <outfile>
insert into index   <attr, value, attr, value, . . .>
update index  <attr, value, attr, value, . . .>
delete from index
delete rectangle    <id>
first in index
next in index
last in index
previous in index
fetch from index
batch search index  <infile> <idfile> [<tuplefile>]
search index  <idfile> [<tuplefile>]
list index
retrieve index <indexname> <indexset> <mode> [<tag>]
save index    <tag>
return index  <tag>
pick index    <tag>
build boolean  <boolean definition> [<tag>]
list boolean
pick boolean   <tag>
modify boolean <boolean definition>
drop boolean  <tag>
build select  <attr, attr, . . .> [<tag>]
list select
pick select   <tag>
modify select <attr, attr, . . .>
drop select   <tag>
create indexset     <indexset>
delete indexset     <indexset>
>>> COMMAND EXECUTED: create indexset sarahiset
RETURN CODE: 0
>>> COMMAND EXECUTED: create indexset aliset
RETURN CODE: 0
>>> COMMAND EXECUTED: create indexset jeniset
RETURN CODE: 0
>>> COMMAND EXECUTED: create indexset joiset
RETURN CODE: 0
>>> COMMAND EXECUTED: create index indexhp sarahiset heap ../data/heap.def
RETURN CODE: 0
>>> COMMAND EXECUTED: create index indexhs sarahiset hash ../data/hash.def
RETURN CODE: 0
>>> COMMAND EXECUTED: create index indexbt sarahiset btree ../data/btree.def
```

```
RETURN CODE: 0
>>> COMMAND EXECUTED: create index indexrt sarahiset rtree ../data/rtree.def
RETURN CODE: 0
>>> COMMAND EXECUTED: load index indexhp sarahiset ../data/sarahiset.data
RETURN CODE: 0
>>> COMMAND EXECUTED: load index indexhs sarahiset ../data/aliset.data
RETURN CODE: 0
>>> COMMAND EXECUTED: load index indexbt sarahiset ../data/jeniset.data
RETURN CODE: 0
>>> COMMAND EXECUTED: load index indexrt sarahiset ../data/rtree.data
RETURN CODE: 0
>>> COMMAND EXECUTED: unload index indexhp sarahiset indexhp.unload
RETURN CODE: 0
>>> COMMAND EXECUTED: unload index indexhs sarahiset indexhs.unload
RETURN CODE: 0
>>> COMMAND EXECUTED: unload index indexbt sarahiset indexbt.unload
RETURN CODE: 0
>>> COMMAND EXECUTED: unload index indexrt sarahiset indexrt.unload
RETURN CODE: 0
>>> COMMAND EXECUTED: help index indexhp sarahiset indexhp.help
RETURN CODE: 0
>>> COMMAND EXECUTED: help index indexhs sarahiset indexhs.help
RETURN CODE: 0
>>> COMMAND EXECUTED: help index indexbt sarahiset indexbt.help
RETURN CODE: 0
>>> COMMAND EXECUTED: help index indexrt sarahiset indexrt.help
RETURN CODE: 0
>>> COMMAND EXECUTED: quit
RETURN CODE: 0
```

```
                                    TEST 2
Please enter your username: Please enter your password:
Need to validate user and password.
COMMANDS AND THEIR SYNTAX:

create index  <indexname> <indexset> <format> <infile>
drop index    <indexname> <indexset>
copy index    <from-index> <from-indexset> <to-index> <to-indexset> [<format>]
move index    <from-index> <from-indexset> <to-index> <to-indexset>
help index    <indexname> <indexset>
load index    <indexname> <indexset> <infile>
unload index  <indexname> <indexset> <outfile>
insert into index   <attr, value, attr, value, . . .>
update index  <attr, value, attr, value, . . .>
delete from index
delete rectangle    <id>
first in index
next in index
last in index
previous in index
fetch from index
batch search index  <infile> <idfile> [<tuplefile>]
search index  <idfile> [<tuplefile>]
list index
retrieve index <indexname> <indexset> <mode> [<tag>]
save index    <tag>
return index  <tag>
pick index    <tag>
build boolean  <boolean definition> [<tag>]
list boolean
pick boolean   <tag>
modify boolean <boolean definition>
drop boolean  <tag>
build select  <attr, attr, . . .> [<tag>]
list select
pick select   <tag>
modify select <attr, attr, . . .>
drop select   <tag>
create indexset     <indexset>
delete indexset     <indexset>
>>> COMMAND EXECUTED: copy index indexhp sarahiset indexhp aliset
RETURN CODE: 0
>>> COMMAND EXECUTED: copy index indexhs sarahiset indexhs aliset
RETURN CODE: 0
>>> COMMAND EXECUTED: copy index indexbt sarahiset indexbt aliset
RETURN CODE: 0
>>> COMMAND EXECUTED: copy index indexrt sarahiset indexrt aliset
RETURN CODE: 0
>>> COMMAND EXECUTED: copy index indexhp sarahiset indexhs1 jeniset hash
RETURN CODE: 0
>>> COMMAND EXECUTED: copy index indexhp sarahiset indexbt1 jeniset btree
RETURN CODE: 0
>>> COMMAND EXECUTED: copy index indexhs sarahiset indexhp1 jeniset heap
```

```
RETURN CODE: 0
>>> COMMAND EXECUTED: copy index indexhs sarahiset indexbt2 jeniset btree
RETURN CODE: 0
>>> COMMAND EXECUTED: copy index indexbt sarahiset indexhp2 jeniset heap
RETURN CODE: 0
>>> COMMAND EXECUTED: copy index indexbt sarahiset indexhs2 jeniset hash
RETURN CODE: 0
>>> COMMAND EXECUTED: move index indexhp sarahiset indexhp joiset
RETURN CODE: 0
>>> COMMAND EXECUTED: move index indexhs sarahiset indexhs joiset
RETURN CODE: 0
>>> COMMAND EXECUTED: move index indexbt sarahiset indexbt joiset
RETURN CODE: 0
>>> COMMAND EXECUTED: move index indexrt sarahiset indexrt joiset
RETURN CODE: 0
>>> COMMAND EXECUTED: quit
RETURN CODE: 0
```

TEST 3

```
Please enter your username: Please enter your password:
Need to validate user and password.
COMMANDS AND THEIR SYNTAX:

create index  <indexname> <indexset> <format> <infile>
drop index    <indexname> <indexset>
copy index    <from-index> <from-indexset> <to-index> <to-indexset> [<format>]
move index    <from-index> <from-indexset> <to-index> <to-indexset>
help index    <indexname> <indexset>
load index    <indexname> <indexset> <infile>
unload index  <indexname> <indexset> <outfile>
insert into index   <attr, value, attr, value, . . .>
update index  <attr, value, attr, value, . . .>
delete from index
delete rectangle    <id>
first in index
next in index
last in index
previous in index
fetch from index
batch search index  <infile> <idfile> [<tuplefile>]
search index  <idfile> [<tuplefile>]
list index
retrieve index <indexname> <indexset> <mode> [<tag>]
save index    <tag>
return index  <tag>
pick index    <tag>
build boolean  <boolean definition> [<tag>]
list boolean
pick boolean   <tag>
modify boolean <boolean definition>
drop boolean  <tag>
build select  <attr, attr, . . .> [<tag>]
list select
pick select   <tag>
modify select <attr, attr, . . .>
drop select   <tag>
create indexset     <indexset>
delete indexset     <indexset>
>>> COMMAND EXECUTED: retrieve index indexhp joiset read_only ihp
RETURN CODE: 0
>>> COMMAND EXECUTED: list index
       Tag: IHP      Value: JOISET/INDEXHP
RETURN CODE: 0
>>> COMMAND EXECUTED: pick index ihp
RETURN CODE: 0
>>> COMMAND EXECUTED: first in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: aloysius yoon|123
RETURN CODE: 0
>>> COMMAND EXECUTED: previous in index
```

```
RETURN CODE: -2
>>> COMMAND EXECUTED: first in index
RETURN CODE: 0
>>> COMMAND EXECUTED: next in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: jennifer carle|234
RETURN CODE: 0
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: barb tower|45
RETURN CODE: 0
>>> COMMAND EXECUTED: next in index
RETURN CODE: -2
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: previous in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: dean perkins|34
RETURN CODE: 0
>>> COMMAND EXECUTED: return index ihp
RETURN CODE: 0
>>> COMMAND EXECUTED: retrieve index indexhs joiset read_only ihs
RETURN CODE: 0
>>> COMMAND EXECUTED: pick index ihs
RETURN CODE: 0
>>> COMMAND EXECUTED: first in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: stephen wallace|5678
RETURN CODE: 0
>>> COMMAND EXECUTED: previous in index
RETURN CODE: -2
>>> COMMAND EXECUTED: first in index
RETURN CODE: 0
>>> COMMAND EXECUTED: next in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: lillian wallace|6789
RETURN CODE: 0
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: richard wallace|2345
RETURN CODE: 0
>>> COMMAND EXECUTED: next in index
RETURN CODE: -2
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: previous in index
RETURN CODE: 0
```

```
>>> COMMAND EXECUTED: fetch from index
Tuple: sandra wallace|1234
RETURN CODE: 0
>>> COMMAND EXECUTED: return index ihs
RETURN CODE: 0
>>> COMMAND EXECUTED: retrieve index indexbt joiset read_only ibt
RETURN CODE: 0
>>> COMMAND EXECUTED: pick index ibt
RETURN CODE: 0
>>> COMMAND EXECUTED: first in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: allen wallace|7890
RETURN CODE: 0
>>> COMMAND EXECUTED: previous in index
RETURN CODE: -2
>>> COMMAND EXECUTED: first in index
RETURN CODE: 0
>>> COMMAND EXECUTED: next in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: amy wallace|12
RETURN CODE: 0
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: wally tower|56
RETURN CODE: 0
>>> COMMAND EXECUTED: next in index
RETURN CODE: -2
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: previous in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: peggy wallace|8901
RETURN CODE: 0
>>> COMMAND EXECUTED: return index ibt
RETURN CODE: 0
>>> COMMAND EXECUTED: retrieve index indexrt joiset read_onlu irt
RETURN CODE: 0
>>> COMMAND EXECUTED: pick index irt
RETURN CODE: 0
>>> COMMAND EXECUTED: batch search index ../data/rtree.search rtree-id.out rtree-tup.out
RETURN CODE: 0
>>> COMMAND EXECUTED: return index irt
RETURN CODE: 0
>>> COMMAND EXECUTED: quit
RETURN CODE: 0
```

```
                                    TEST 4
Please enter your username: Please enter your password:
Need to validate user and password.
COMMANDS AND THEIR SYNTAX:

create index  <indexname> <indexset> <format> <infile>
drop index    <indexname> <indexset>
copy index    <from-index> <from-indexset> <to-index> <to-indexset> [<format>]
move index    <from-index> <from-indexset> <to-index> <to-indexset>
help index    <indexname> <indexset>
load index    <indexname> <indexset> <infile>
unload index  <indexname> <indexset> <outfile>
insert into index   <attr, value, attr, value, . . .>
update index  <attr, value, attr, value, . . .>
delete from index
delete rectangle    <id>
first in index
next in index
last in index
previous in index
fetch from index
batch search index  <infile> <idfile> [<tuplefile>]
search index  <idfile> [<tuplefile>]
list index
retrieve index <indexname> <indexset> <mode> [<tag>]
save index    <tag>
return index  <tag>
pick index    <tag>
build boolean  <boolean definition> [<tag>]
list boolean
pick boolean   <tag>
modify boolean <boolean definition>
drop boolean  <tag>
build select  <attr, attr, . . .> [<tag>]
list select
pick select   <tag>
modify select <attr, attr, . . .>
drop select   <tag>
create indexset     <indexset>
delete indexset     <indexset>
>>> COMMAND EXECUTED: retrieve index indexhp joiset read_only ihp
RETURN CODE: 0
>>> COMMAND EXECUTED: list index
       Tag: IHP      Value: JOISET/INDEXHP
RETURN CODE: 0
>>> COMMAND EXECUTED: pick index ihp
RETURN CODE: 0
>>> COMMAND EXECUTED: build boolean attra < "barbara" ib1
RETURN CODE: 0
>>> COMMAND EXECUTED: pick boolean ib1
RETURN CODE: 0
>>> COMMAND EXECUTED: first in index
RETURN CODE: 0
```

```
>>> COMMAND EXECUTED: fetch from index
Tuple: aloysius yoon|123
RETURN CODE: 0
>>> COMMAND EXECUTED: previous in index
RETURN CODE: -2
>>> COMMAND EXECUTED: first in index
RETURN CODE: 0
>>> COMMAND EXECUTED: next in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: allen wallace|7890
RETURN CODE: 0
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: barb tower|45
RETURN CODE: 0
>>> COMMAND EXECUTED: next in index
RETURN CODE: -2
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: previous in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: amy wallace|12
RETURN CODE: 0
>>> COMMAND EXECUTED: return index ihp
RETURN CODE: 0
>>> COMMAND EXECUTED: retrieve index indexhs joiset read_only ihs
RETURN CODE: 0
>>> COMMAND EXECUTED: pick index ihs
RETURN CODE: 0
>>> COMMAND EXECUTED: pick boolean ib1
RETURN CODE: 0
>>> COMMAND EXECUTED: first in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: amy wallace|12
RETURN CODE: 0
>>> COMMAND EXECUTED: previous in index
RETURN CODE: -2
>>> COMMAND EXECUTED: first in index
RETURN CODE: 0
>>> COMMAND EXECUTED: next in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: allen wallace|7890
RETURN CODE: 0
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: allen wallace|7890
RETURN CODE: 0
```

```
>>> COMMAND EXECUTED: next in index
RETURN CODE: -2
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: previous in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: amy wallace|12
RETURN CODE: 0
>>> COMMAND EXECUTED: return index ihs
RETURN CODE: 0
>>> COMMAND EXECUTED: retrieve index indexbt joiset read_only ibt
RETURN CODE: 0
>>> COMMAND EXECUTED: pick index ibt
RETURN CODE: 0
>>> COMMAND EXECUTED: pick boolean ib1
RETURN CODE: 0
>>> COMMAND EXECUTED: first in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: allen wallace|7890
RETURN CODE: 0
>>> COMMAND EXECUTED: previous in index
RETURN CODE: -2
>>> COMMAND EXECUTED: first in index
RETURN CODE: 0
>>> COMMAND EXECUTED: next in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: amy wallace|12
RETURN CODE: 0
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: barb tower|45
RETURN CODE: 0
>>> COMMAND EXECUTED: next in index
RETURN CODE: -2
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: previous in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: amy wallace|12
RETURN CODE: 0
>>> COMMAND EXECUTED: return index ibt
RETURN CODE: 0
>>> COMMAND EXECUTED: drop boolean ib1
RETURN CODE: 0
>>> COMMAND EXECUTED: quit
RETURN CODE: 0
```

TEST 5

```
Please enter your username: Please enter your password:
Need to validate user and password.
COMMANDS AND THEIR SYNTAX:

create index  <indexname> <indexset> <format> <infile>
drop index    <indexname> <indexset>
copy index    <from-index> <from-indexset> <to-index> <to-indexset> [<format>]
move index    <from-index> <from-indexset> <to-index> <to-indexset>
help index    <indexname> <indexset>
load index    <indexname> <indexset> <infile>
unload index  <indexname> <indexset> <outfile>
insert into index   <attr, value, attr, value, . . .>
update index  <attr, value, attr, value, . . .>
delete from index
delete rectangle    <id>
first in index
next in index
last in index
previous in index
fetch from index
batch search index  <infile> <idfile> [<tuplefile>]
search index  <idfile> [<tuplefile>]
list index
retrieve index <indexname> <indexset> <mode> [<tag>]
save index    <tag>
return index  <tag>
pick index    <tag>
build boolean  <boolean definition> [<tag>]
list boolean
pick boolean   <tag>
modify boolean <boolean definition>
drop boolean  <tag>
build select  <attr, attr, . . .> [<tag>]
list select
pick select   <tag>
modify select <attr, attr, . . .>
drop select   <tag>
create indexset     <indexset>
delete indexset     <indexset>
>>> COMMAND EXECUTED: retrieve index indexhp joiset modify ihp
RETURN CODE: 0
>>> COMMAND EXECUTED: list index
       Tag: IHP      Value: JOISET/INDEXHP
RETURN CODE: 0
>>> COMMAND EXECUTED: pick index ihp
RETURN CODE: 0
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: barb tower|45
RETURN CODE: 0
>>> COMMAND EXECUTED: update index attra|timos sellis|id|3030
```

```
RETURN CODE: 0
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: timos sellis|3030
RETURN CODE: 0
>>> COMMAND EXECUTED: delete from index
RETURN CODE: 0
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: dean perkins|34
RETURN CODE: 0
>>> COMMAND EXECUTED: insert into index attra|nick roussopoulous|id|9090
RETURN CODE: 0
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: nick roussopoulous|9090
RETURN CODE: 0
>>> COMMAND EXECUTED: save index ihp
RETURN CODE: 0
>>> COMMAND EXECUTED: return index ihp
RETURN CODE: 0
>>> COMMAND EXECUTED: retrieve index indexhs joiset modify ihs
RETURN CODE: 0
>>> COMMAND EXECUTED: pick index ihs
RETURN CODE: 0
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: richard wallace|2345
RETURN CODE: 0
>>> COMMAND EXECUTED: update index attra|richard wallace|id|3030
RETURN CODE: 0
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: richard wallace|3030
RETURN CODE: 0
>>> COMMAND EXECUTED: delete from index
RETURN CODE: 0
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: sandra wallace|1234
RETURN CODE: 0
>>> COMMAND EXECUTED: insert into index attra|nick roussopoulous|id|9090
RETURN CODE: 0
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: sandra wallace|1234
```

```
RETURN CODE: 0
>>> COMMAND EXECUTED: save index ihs
RETURN CODE: 0
>>> COMMAND EXECUTED: return index ihs
RETURN CODE: 0
>>> COMMAND EXECUTED: retrieve index indexbt joiset modify ibt
RETURN CODE: 0
>>> COMMAND EXECUTED: pick index ibt
RETURN CODE: 0
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: wally tower|56
RETURN CODE: 0
>>> COMMAND EXECUTED: update index attra|timos sellis|id|3030
RETURN CODE: 0
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: timos sellis|3030
RETURN CODE: 0
>>> COMMAND EXECUTED: delete from index
RETURN CODE: 0
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: peggy wallace|8901
RETURN CODE: 0
>>> COMMAND EXECUTED: insert into index attra|nick roussopoulous|id|9090
RETURN CODE: 0
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: peggy wallace|8901
RETURN CODE: 0
>>> COMMAND EXECUTED: save index ibt
RETURN CODE: 0
>>> COMMAND EXECUTED: return index ibt
RETURN CODE: 0
>>> COMMAND EXECUTED: retrieve index indexhp joiset read_only ihp
RETURN CODE: 0
>>> COMMAND EXECUTED: list index
       Tag: IHP      Value: JOISET/INDEXHP
RETURN CODE: 0
>>> COMMAND EXECUTED: pick index ihp
RETURN CODE: 0
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: nick roussopoulous|9090
RETURN CODE: 0
>>> COMMAND EXECUTED: update index attra|timos sellis|id|3030
RETURN CODE: -9
```

```
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: nick roussopoulous|9090
RETURN CODE: 0
>>> COMMAND EXECUTED: delete from index
RETURN CODE: -9
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: nick roussopoulous|9090
RETURN CODE: 0
>>> COMMAND EXECUTED: insert into index attra|nick roussopoulous|id|9090
RETURN CODE: -9
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: nick roussopoulous|9090
RETURN CODE: 0
>>> COMMAND EXECUTED: save index ihp
RETURN CODE: -9
>>> COMMAND EXECUTED: return index ihp
RETURN CODE: 0
>>> COMMAND EXECUTED: retrieve index indexhs joiset read_only ihs
RETURN CODE: 0
>>> COMMAND EXECUTED: pick index ihs
RETURN CODE: 0
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: sandra wallace|1234
RETURN CODE: 0
>>> COMMAND EXECUTED: update index attra|timos sellis|id|3030
RETURN CODE: -9
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: sandra wallace|1234
RETURN CODE: 0
>>> COMMAND EXECUTED: delete from index
RETURN CODE: -9
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: sandra wallace|1234
RETURN CODE: 0
>>> COMMAND EXECUTED: insert into index attra|nick roussopoulous|id|9090
RETURN CODE: -9
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: sandra wallace|1234
RETURN CODE: 0
```

```
>>> COMMAND EXECUTED: save index ihs
RETURN CODE: -9
>>> COMMAND EXECUTED: return index ihs
RETURN CODE: 0
>>> COMMAND EXECUTED: retrieve index indexbt joiset read_only ibt
RETURN CODE: 0
>>> COMMAND EXECUTED: pick index ibt
RETURN CODE: 0
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: peggy wallace|8901
RETURN CODE: 0
>>> COMMAND EXECUTED: update index attra|timos sellis|id|3030
RETURN CODE: -9
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: peggy wallace|8901
RETURN CODE: 0
>>> COMMAND EXECUTED: delete from index
RETURN CODE: -9
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: peggy wallace|8901
RETURN CODE: 0
>>> COMMAND EXECUTED: insert into index attra|nick roussopoulous|id|9090
RETURN CODE: -9
>>> COMMAND EXECUTED: last in index
RETURN CODE: 0
>>> COMMAND EXECUTED: fetch from index
Tuple: peggy wallace|8901
RETURN CODE: 0
>>> COMMAND EXECUTED: save index ibt
RETURN CODE: -9
>>> COMMAND EXECUTED: return index ibt
RETURN CODE: 0
>>> COMMAND EXECUTED: quit
RETURN CODE: 0
```

```
                                    TEST 6
Please enter your username: Please enter your password:
Need to validate user and password.
COMMANDS AND THEIR SYNTAX:

create index  <indexname> <indexset> <format> <infile>
drop index    <indexname> <indexset>
copy index    <from-index> <from-indexset> <to-index> <to-indexset> [<format>]
move index    <from-index> <from-indexset> <to-index> <to-indexset>
help index    <indexname> <indexset>
load index    <indexname> <indexset> <infile>
unload index  <indexname> <indexset> <outfile>
insert into index   <attr, value, attr, value, . . .>
update index  <attr, value, attr, value, . . .>
delete from index
delete rectangle    <id>
first in index
next in index
last in index
previous in index
fetch from index
batch search index  <infile> <idfile> [<tuplefile>]
search index  <idfile> [<tuplefile>]
list index
retrieve index <indexname> <indexset> <mode> [<tag>]
save index    <tag>
return index  <tag>
pick index    <tag>
build boolean  <boolean definition> [<tag>]
list boolean
pick boolean   <tag>
modify boolean <boolean definition>
drop boolean  <tag>
build select  <attr, attr, . . .> [<tag>]
list select
pick select   <tag>
modify select <attr, attr, . . .>
drop select   <tag>
create indexset     <indexset>
delete indexset     <indexset>
>>> COMMAND EXECUTED: drop index indexhp aliset
RETURN CODE: 0
>>> COMMAND EXECUTED: drop index indexhs aliset
RETURN CODE: 0
>>> COMMAND EXECUTED: drop index indexbt aliset
RETURN CODE: 0
>>> COMMAND EXECUTED: drop index indexhp joiset
RETURN CODE: 0
>>> COMMAND EXECUTED: drop index indexhs joiset
RETURN CODE: 0
>>> COMMAND EXECUTED: drop index indexbt joiset
RETURN CODE: 0
>>> COMMAND EXECUTED: drop index indexhp1 jeniset
```

```
RETURN CODE: 0
>>> COMMAND EXECUTED: drop index indexhp2 jeniset
RETURN CODE: 0
>>> COMMAND EXECUTED: drop index indexhs1 jeniset
RETURN CODE: 0
>>> COMMAND EXECUTED: drop index indexhs2 jeniset
RETURN CODE: 0
>>> COMMAND EXECUTED: drop index indexbt1 jeniset
RETURN CODE: 0
>>> COMMAND EXECUTED: drop index indexbt2 jeniset
RETURN CODE: 0
>>> COMMAND EXECUTED: delete indexset sarahiset
RETURN CODE: 0
>>> COMMAND EXECUTED: delete indexset aliset
RETURN CODE: 0
>>> COMMAND EXECUTED: delete indexset jeniset
RETURN CODE: 0
>>> COMMAND EXECUTED: delete indexset joiset
RETURN CODE: 0
>>> COMMAND EXECUTED: quit
RETURN CODE: 0
```

**NASA** Report Documentation Page

| | | |
|---|---|---|
| 1. Report No.<br>FINAL | 2. Government Assession No. | 3. Recipient's Catalog No. |
| 4. Title and Subtitle<br><br>VIEWCACHE: An Incremental Pointer-Based Access Method for Distributed Databases | | 5. Report Date<br><br>31-Oct-92<br>6. Performing Organization Code<br>N/A |
| 7. Author(s)<br><br>Steve Kelley, Nick Roussopoulos, Timos Sellis | | 8. Performing Organization Report No.<br>FINAL<br>10. Work Unit No. |
| 9. Performing Organization Name and Address<br><br>Advanced Communication Technology Inc.<br>1209 Goth Lane<br>Silver Spring, MD 20905 | | 11. Contract or Grant No.<br><br>NAS5-30628 |
| 12. Sponsoring Agency Name and Address<br>NASA<br>Washington, D.C. 20546-0001<br><br>TO: Dr. Barry Jacobs<br>Code 934<br>Goddard Space Flight Center<br>Greenbelt, MD 20771 | | 13. Type of Report and Period Covered<br><br>FINAL<br>May 2, 1989- October 31, 1992<br>14. Sponsoring Agency Code |

15. Supplementary Notes

16. Abstract

The goal of the Universal Index System (UIS), is to provide an easy-to-use and reliable interface to many different kinds of database systems. The impetus for this system was to simplify database index management for users, thus encouraging the use of indexes. As the idea grew into an actual system design, the concept of increasing database performance by facilitating the use of time-saving techniques at the user level became a theme for the project. were don on the IUE database.
This Final Report describes the Design, the Implementation of UIS, and its Language Interfaces. It also includes the User's Guide and the Reference Manual.

| | | | |
|---|---|---|---|
| 17. Key Words (Suggested by Author(s))<br><br>Indexing, Spatial Access Methods, R-trees, Index Management System | 18. Distribution Statement | | |
| 19. Security Classif. (of this report)<br>U | 20. Security Classif. (of page)<br>U | 21. No. of Pages<br>563 | 22. Price |

NASA FORM 1626 Oct 86